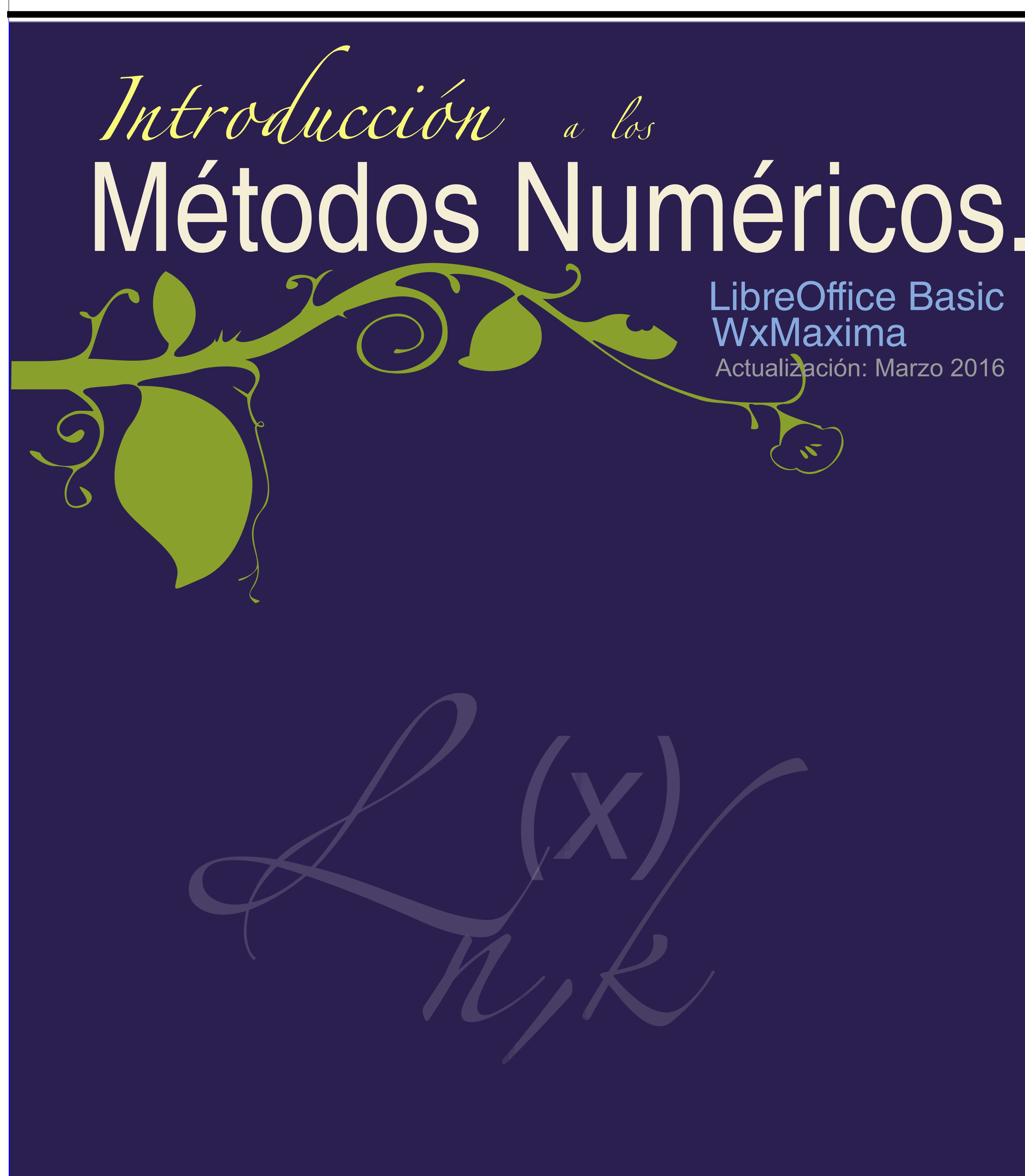
Walter Mora F.
Escuela de Matemática
Instituto Tecnológico de Costa Rica



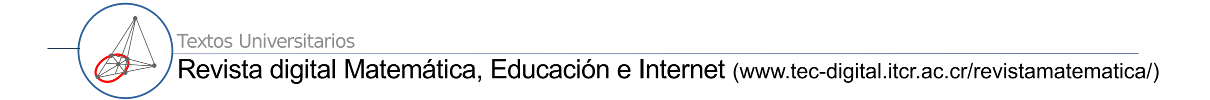
Revista digital Matemática, Educación e Internet www.tec-digital.itcr.ac.cr/revistamatematica/

INTRODUCCIÓN a los MÉTODOS NUMÉRICOS.

Implementaciones en Basic (LibreOffice, Excel) y wxMaxima. Actualización parcial: Marzo 2016.

Prof. Walter Mora F.,
Escuela de Matemática
Instituto Tecnológico de Costa Rica.
(www.tec-digital.itcr.ac.cr/revistamatematica/)





Correo Electrónico: wmora2@gmail.com
Escuela de Matemática
Instituto Tecnológico de Costa Rica
Apdo. 159-7050, Cartago
Teléfono (506)25502225
Fax (506)25502493

Mora Flores, Walter.
Introducción a los métodos numéricos. Implementaciones en
Basic-Calc de Libre
Office y wxMaxima
. 1ra ed.
– Escuela de Matemática,Instituto Tecnológico de Costa Rica. 2010.
309 pp.
ISBN Obra Independiente: 978-9968-641-13-5
1. Métodos Numéricos. 2. Programación 3. Algoritmos.

Contenido

Prefacio

El propósito de este libro es la implementación de métodos numéricos básicos usando la hoja electrónica de LibreOffice (ver apéndice A). Por supuesto, también se puede usar con Calc de OpenOffice y, con algunos cambios pequeños, se puede usar con VBA Excel. También se hace implementaciones con wxMaxima y a veces con Wolfram Mathematica. El curso esta orientado a estudiantes con poco o casi ningún conocimiento de programación. Usar la hoja electrónica es ventajoso pues permite tener los datos en un ambiente natural y conocido y aplicar métodos numéricos con una macro programando en un lenguaje sencillo, natural y muy amigable (lo cual no siempre es bueno). Sin embargo, estas hojas electrónicas tienen sus limitaciones y, en algun momento se debe pasar a usar software simbólico, digamos wxMaxima u Octave. También se podría usar software privativo como MatLab o Mathematica pero eso conlleva el problema de las licencias. En este libro se expone la teoría, a veces con justificaciones teóricas, se presentan varios ejemplos y al final del cada capítulo, se dedica tiempo a los algoritmos y las implementaciones.

W. MORA.

Cartago, Costa Rica

Julio, 2010.

1 ARITMÉTICA DEL COMPUTADOR Y ERRORES

1.1 Introducción

Hay varias fuentes de error en los cálculos, aquí nos interesa los errores de truncación inicial y errores de redondeo. Los cálculos numéricos son llevados a cabo con un número finito de dígitos aunque la mayoría de números tengan un número infinito de dígitos. Cuando los números son muy grandes o muy pequeños se representan con *punto flotante*, por ejemplo

$$x = p \times 10^q$$

donde p es un número cercano a 1 y q es un entero; por ejemplo $0.00147 = 0.147 \times 10^{-2}$.

Cuando contamos el número de dígitos en un valor numérico, las *cifras significativas* son los dígitos usados para expresar el número: 1,2,3,4,5,6,7,8 y 9. El cero es también signicativo, excepto cuando es usado para para fijar el punto decimal o para llenar los lugares de dígitos desconocidos o descartados. Por ejemplo, 0.147 y 3.23 tienen tres cifras significativas mientras que $0.000207 = 0.207 \times 10^{-3}$ tiene tres cifras significativas.

Decimales correctos y dígitos significativos en una aproximación. Si $\tilde{p}$ *es una aproximación* de p (digamos que con más de k decimales) tal que $|\tilde{p} - p|$ *no excede* 0.5×10^{-k}, decimos que $\tilde{p}$ tiene k *decimales correctos,* es decir, k es el natural más grande tal que $|\tilde{p} - p| \leq 0.5 \times 10^{-k}$.

Ejemplo 1.1

Por ejemplo, si $p = 0.001234$ y $\tilde{p} = 0.001234 \pm 0.000004$, entonces $|\tilde{p} - p| \leq 0.5 \times 10^{-5}$ (pero $|\tilde{p} - p| \not\leq 0.5 \times 10^{-6}$) así que $\tilde{p}$ tiene 5 decimales correctos. En efecto, $0.001234 + 0.000004 = 0.001238$ y $0.001234 - 0.000004 = 0.00123$

Si $\tilde{p}$ tiene k decimales correctos, los *dígitos* en $\tilde{p}$ que ocupan posiciones donde "la unidad" es $\geq 10^{-k}$ se llaman *dígitos significativos* (es decir, los ceros iniciales no cuentan).

Ejemplo 1.2

Por ejemplo, si $\tilde{p} = 0.001234$ aproxima p con 5 decimales correctos, entonces $\tilde{p}$ tiene tres dígitos significativos pues 0.001, 0.0002, 0.00003 son $> 10^{-5}$ pero $0.000004 \not> 10^{-5}$.

Ejemplo 1.3

Si sabemos que $\tilde{p} = 1.5756457865$ es una aproximación de un número desconocido p y que $|\,p - \tilde{p}\,| \leq 0.000000005$, entonces al menos sabemos que $|\,p - \tilde{p}\,| \leq 0.5 \times 10^{-7}$ (pero *no* sabemos si $|\,p - \tilde{p}\,| \leq 0.5 \times 10^{-8}$) con lo que $\tilde{p}$ tiene *como mínimo*, 7 decimales correctos.

Error absoluto y error realtivo. Sea $\tilde{p}$ es una aproximación de p, entonces,

- $|\,\tilde{p} - p\,|$ es el *error absoluto* de la aproximación,

- $\dfrac{|\tilde{p} - p|}{|p\,|}$, $p \neq 0$; es el *error relativo* de la aproximación.

El error relativo nos da un porcentaje del error (si lo multiplicamos por 100). En general, el error relativo es más significativo que el error absoluto si tratamos con números muy pequeños o números muy grandes.

Ejemplo 1.4

Error relativo y error absoluto para diferentes valores.

p	$\tilde{p}$	Error absoluto	Error relativo (%)
0.000012	0.000009	0.000003	25%
4.5×10^{-7}	6.0×10^{-11}	4.4994×10^{-7}	99.98667%
10000000	1000000	9000000	90%
0.123456	0.1234	0.000056	0.0453603%

Para determinar el error absoluto y el error relativo, deberíamos conocer el valor exacto y el valor aproximado. En la práctica de los métodos numéricos, no conocemos *a priori* el valor exacto que estamos buscando pero si algunas aproximaciones. A veces tenemos una aproximación del error absoluto o a veces solo unas cuantas aproximaciones; en este último caso, la estimación del error relativo se hace usando la mejor estimación disponible de p.

Ejemplo 1.5

Sea $x_0 = 1$ y $x_{n+1} = \dfrac{1}{2}\left(x_n + \dfrac{2}{x_n}\right)$. Se sabe que $\lim\limits_{n\to\infty} x_n = \sqrt{2}$. Podemos aproximar $\sqrt{2}$ usando esta sucesión. La estimación del error relativo se hace *usando a* x_{n+1} *como la mejor estimación del valor exacto* (pues la sucesión es decreciente). En la tabla que sigue, comparamos la estimación del error relativo con el error relativo exacto.

	Error relativo estimado	Error relativo exacto
$x_0 = 1$	$\lvert x_n - x_{n+1}\rvert / \lvert x_{n+1}\rvert$	$\lvert\sqrt{2} - x_{n+1}\rvert / \lvert\sqrt{2}\rvert$
$x_1 = 1.5.$	0.333333	0.0606602
$x_2 = 1.4166666666666666667$	0.0588235	0.00173461
$x_3 = 1.4142156862745098039$	0.0017331	$1.5018250929351827 \times 10^{-6}$
$x_4 = 1.4142135623746899106$	$1.5018239652057424 \times 10^{-6}$	$1.1276404038266872 \times 10^{-12}$
$x_5 = 1.4142135623730950488$	$1.1277376112344212 \times 10^{-12}$	0.

Usando el error relativo estimado podemos decir que $\sqrt{2} \approx 1.4142135623730950488$ con error relativo $\leq 1.1277376112344212 \times 10^{-12}$.

EJERCICIOS

1.1 si $p = 0.001234$ y $\tilde{p} = 0.001234 \pm 0.000006$, verifique que $\tilde{p}$ tiene 4 decimales correctos y solo dos dígitos significativos.

1.2 Si se sabe que $\tilde{p} = 4.6565434$ aproxima a un número p con $\lvert\, p - \tilde{p}\,\rvert \leq 0.000001$, entonces como mínimo ¿cuántos decimales exactos habría en la aproximación?

1.3 Encuentre un valor de tolerancia δ tal que si $\lvert\, p - \tilde{p}\,\rvert \leq \delta$, podamos garantizar que $\tilde{p}$ tendrá al menos tres decimales exactos.

1.2 Aritmética del computador.

La representación de un número real en coma flotante requiere una base β y una precisión p. Por ejemplo, si $\beta = 10$ y $p = 3$ entonces 0.1 se representa como 1.00×10^{-1}.

El número

$$\pm\left(d_0 + \frac{d_1}{\beta^1} + \frac{d_2}{\beta^2} + ... + \frac{d_{p-1}}{\beta^{p-1}}\right) \times \beta^e,\ \ 0 \leq d_i \leq \beta - 1 \tag{1.1}$$

se representa en coma flotante como

$$\pm\, d_0.d_1d_2....d_{p-1} \times \beta^e,\ \ 0 \leq d_i < \beta \tag{1.2}$$

$d_0.d_1d_2....d_{p-1}$ se llama “mantisa” y tiene p dígitos.

Ejemplo 1.6

En base 10

$$3213 = \left(\frac{3}{10}+\frac{2}{10^2}+\frac{1}{10^3}+\frac{3}{10^4}\right)\times 10^4 = 0.3213\times 10^4$$
$$= 0.03213\times 10^5$$

Ejemplo 1.7

$$\begin{aligned} 3.125 &= 2^0+2^1+\frac{1}{2^3} &&= (11.001)_2\times 2^0 \\ &= \left(2+\frac{1}{2^1}+\frac{1}{2^4}\right)\times 2^1 &&= (1.1001)_2\times 2^1 \\ &= \left(\frac{1}{2}+\frac{1}{2^2}+\frac{1}{2^5}\right)\times 2^2 &&= (0.11001)_2\times 2^2 \end{aligned}$$

Representación en el computador. Si en un computador los números reales son representados en base 2, estos números se pueden almacenar como una sucesión de bits. Por ejemplo $3.125 = 1.1001\times 2^1$ se almacena en 64 bits como

$$|\underbrace{0}_{\text{Signo}}|\underbrace{10000000000}_{\text{Exponente, 11 bits}}|\underbrace{1100100000000000000000000000000000000000000000000000}_{\text{Mantisa, 52 bits}}| \quad (1.3)$$

Cuente con 15 decimales. Para un número representado en 64 bits, hay 11 bits para el exponente (± 308 en base 10) y 52 bits para la mantisa, esto nos da entre 15 y 17 dígitos decimales de precisión. e_{min} y e_{max} indican el exponente mínimo y el exponente máximo.

Cuando usamos la aritmética del computador, un número real se representa en doble precisión (double) con 15 decimales.

La comparación debe ser sencilla. Aunque los exponentes pueden ser negativos, se introduce un sesgo de tal manera que siempre queden como números mayores o iguales a cero (así, el exponente negativo más pequeño pasa a ser cero y los otros exponentes pasan a ser positivos). Esto se hace con el propósito de que la comparación de los números sea sencilla. En 64 bits el sesgo es 1023, Esta es la razón por la que en (1.3) el exponente es $(10000000000)_2 = 1023+1$.

Cómo lograr la unicidad. La representación en coma flotante no es única. Por ejemplo, 0.1 se puede representar como 0.01×10^1 o como 1.00×10^{-1}. Si en (1.2) pedimos que $d_0 \neq 0$, la representación se dice *normalizada.* La representación de 0.1 como 1.00×10^{-1} es la representación normalizada.

La representación normalizada resuelve la unicidad, pero hay un problema con el cero. Esto se resuelve con una convención para representarlo: $1.0\times e^{e_{min}-1}$. Esto nos resta un exponente en la representación de los números pues

$e_{min} - 1$ se reserva para el cero.

Los números que podemos representar. En la representación normalizada en 64 bits, se sobrentiende que $d_0 = 1$, lo que nos deja 53 bits para la mantisa.

$$x = (-1)^s 1.b_1 b_2 ... b_{52} \times 2^e \quad \text{con} \quad -1022 \leq e \leq 1023$$

El valor más grande sería $1.11...11 \times 2^{1023} \approx 1.79 \times 10^{308}$ y el valor (normal[1]) más pequeño sería $2.22... \times 10^{-308}$.

Excepciones. Si los cálculos exceden el máximo valor representable caemos en un estado de "sobreflujo" (overflow). En el caso del mínimo valor caemos en un estado de "subflujo" (underflow). En el estándar IEEE754, se reservan algunos patrones de bits para " excepciones". Hay un patrón de bits para números que exceden el máximo número representable: `Inf`, `−Inf`, `NaN` (not a number). Por ejemplo, $1./0.$ produce `Inf`. Si un cero es producido por un subflujo debido a negativos muy pequeños, se produce -0 y $1./(-0.)$ produce `− Inf`. `NaN` es producido por cosas tales como $0./0.$, $\sqrt{-2}$ o `Inf − Inf`.

Redondeo. Muchos números racionales tienen un cantidad infinita de decimales en su representación en base 10 y en base 2. Por ejemplo $1/3 = 0.3333333\ldots$. Esto también sucede en la representación en base 2. Por ejemplo,

$$0.2 = (0.00110011001100110...)_2.$$

Se debe hacer un corte para representar estos números en el computador. Si $rd(x)$ es la representación de x (por redondeo simétrico) en doble precisión, entonces

$$\left| \frac{x - \mathrm{rd}(x)}{x} \right| \leq 2^{-52} \leqslant 0.5 \times 10^{-15}$$

1.3 Cancelación

La cancelación ocurre cuando se hace sustracción de dos números muy cercanos. Cuando se forma la resta $a - b$, se representan con el mismo exponente q y algunos dígitos significativos en la mantisa son cancelados y al normalizar se mueven dígitos a la izquierda y se disminuye el exponente. Al final de la mantisa aparecen ceros inútiles.

[1]En aritmética de punto flotante, los números que son más pequeños que el más pequeño número 'normal' que se puede representar, se llaman "subnormales". Los números subnormales se introducen para preservar la importante propiedad $x = y \iff x - y = 0$. Cuando se alcanza el mínimo normal, la mantisa se va rellenando con ceros permitiendo la representación de números más pequeños. El subnormal mínimo sería 4.9×10^{-324}.

Ejemplo 1.8

Por ejemplo, usando aritmética con diez decimales, si $a = \sqrt{9876} = 9.937806599 \times 10^1$ y $b = \sqrt{9875} = 9.937303457 \times 10^1$, entonces $a - b = 0.000503142 \times 10^1$. Normalizando se obtiene,

$$\sqrt{9876} - \sqrt{9875} = 5.031420000 \times 10^{-3}$$

y perdimos dígitos significativos. Se puede arreglar el problema *racionalizando*:

$$\sqrt{a} - \sqrt{b} = \frac{a-b}{\sqrt{a}+\sqrt{b}},$$

esto cambia la resta (de números cercanos) $\sqrt{9876} - \sqrt{9875}$ por la suma $\sqrt{9876} + \sqrt{9875}$ y la resta $9876 - 9875$ no presenta problemas.

$$\sqrt{9876} - \sqrt{9875} = \frac{9876 - 9875}{\sqrt{9876}+\sqrt{9875}} = 5.031418679 \times 10^{-3}.$$

Es usual usar algunos trucos para minimizar este fenómeno de cancelación.

(a.) Cambiar $\sqrt{a} - \sqrt{b}$ por $\dfrac{a-b}{\sqrt{a}+\sqrt{b}}$

(b.) Cambiar $\operatorname{sen} a - \operatorname{sen} b$ por $2\cos\dfrac{a+b}{2}\operatorname{sen}\dfrac{a-b}{2}$

(c.) Cambiar $\log a - \log b$ por $\log(a/b)$

(d.) Si f es suficientemente suave y h pequeño, podemos cambiar $y = f(x+h) - f(x)$ por la expansión de Taylor

$$y = f'(x)h + 0.5f''(x)h^2 + ...$$

Los términos en esta serie decrecen rápidamente si h es suficientemente pequeño, así que el fenómeno de cancelación deja de ser un problema.

A veces, quince dígitos no bastan. Cuando tratamos con números de máquina, hay un error que se va propagando. Esta propagación del error es, en algunos casos, muy dañina. A veces el daño es producto del fenómeno de *cancelación,* al restar números parecidos y muy pequeños y a veces es por problemas de inestabilidad del algoritmo en curso. Algunos cálculos podemos mejorarlos aumentando la precisión, como se muestra en los ejemplos que siguen,

Ejemplo 1.9

Consideremos la cuadrática $P(x) = ax^2 + bx + c$ donde $a = 94906265.625$, $b = -189812534$ y $c = 94906268.375$. Usando Mathematica 8 con la aritmética de la máquina se obtiene

$$x_2 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} = 1.0000000144879793$$

Pero $P(1.0000000144879793) = -2.98023 \times 10^{-8}$!. Mmmmmm, aquí los números que son cercanos son b^2 y $4ac$ y no es de mucha utilidad racionalizar. Todavía nos queda la opción de trabajar con más decimales. En Mathematica 8 aumentamos la precisión, con el código

$$\texttt{SetPrecision[}\frac{-b + \sqrt{b^2 - 4ac}}{2a}\texttt{,16],}$$

y nos devuelve $x_2 = 1.000000028975958$ y ahora, al evaluar nos devuelve $P(1.000000028975958) = 0$.

El comando `NSolve[a*x^2 + b*x + c == 0, x]` devuelve también $x_2 = 1.0000000144879793$, pero también podemos aumentar la precisión con el código (requiere pasar a 'racionales' a, b y c),

```
a = Rationalize[94906265.625];
b = Rationalize[-189812534];
c = Rationalize[94906268.375];
NSolve[a*x^2 + b*x + c == 0, x, WorkingPrecision -> 15],
```

y obtenemos $x_2 = 1.000000028975958$

Ejemplo 1.10

Usando la precisión de la máquina, $\cos(0.00001) = 0.99999999995$. Usando 24 dígitos de precisión, $\cos(0.00001) = 0.999999999949999999586298$.

En Mathematica este cálculo se hace con `SetPrecision[Cos[0.00001],24]`

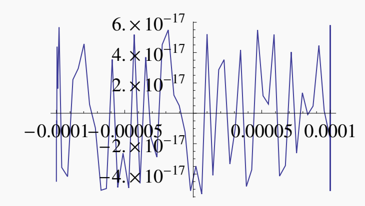
En wxMáxima este cálculo se hace con `block([fpprec:24],bfloat(cos(0.00001)));`

En VBA Excel se debe usar un complemento, por ejemplo XNumbers. *Suponiendo que el complemento está instalado*, el cálculo se podría hacer desde una subrutina con

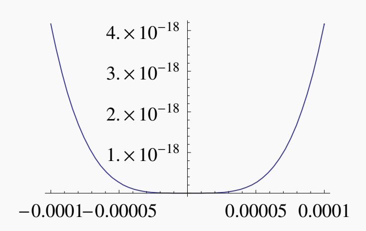
```
Dim mp As New Xnumbers
mp.xCos("0.00001", 24)
```

Ejemplo 1.11

Representación gráfica de $f(x) = x^2/2 + \cos(x) - 1$. La función f siempre es positiva en $[-0.0001, 0.0001]$. Si hacemos la representación gráfica usando números de máquina obtenemos un gráfico inestable,



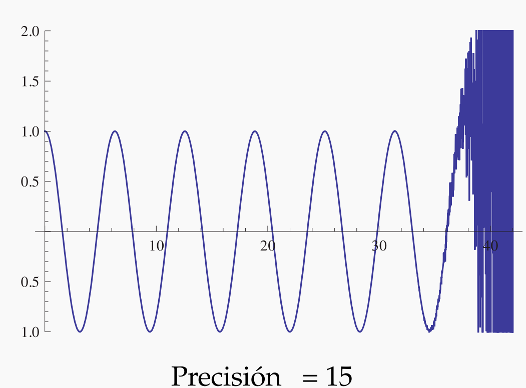
Si aumentamos la precisión a 24 dígitos, la representación gráfica aparece en su forma correcta,



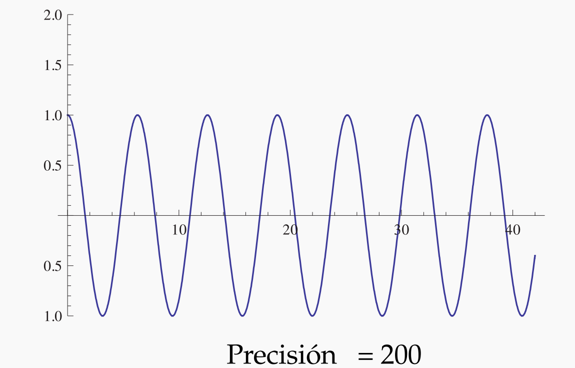
Este último gráfico se generó en Mathematica 6.0 con el código
`Plot[x^2/2 + Cos[x] - 1, {x, -0.0001, 0.0001}, WorkingPrecision -> 24]`. Actualmente (Mathematica 8) controla automáticamente estos problemas de redondeo.

Ejemplo 1.12

Representación gráfica de $f(x) = \sum_{n=0}^{200} \frac{(-1)^n x^{2n}}{(2n)!}$. Aquí también los cálculos se ven afectados si trabajamos con poca precisión.



Precisión = 15



Precisión = 200

Multiprecisión con XNumbers para Excel. XNUMBERS es un complemento que ofrece una gran colección de funciones para Excel, incluyendo multiprecisión (hasta 200 dígitos). Actualmente existe este complemento en versión 'Excel 97/2000/XP/2003' y versión 'Excel2010and2007'. Las instru-cciones y el complemento se puede descargar (agosto 2012) desde `http://www.thetropicalevents.com/Xnumbers60/`. La fuente original de Xnumbers (para Excel 97/2000/XP/2003) es `http://digilander.libero.it/foxes/`.

1.4 Propagación del Error

Cuando un error de redondeo ha sido introducido, este se suma a otros errores y se propaga. Supongamos que queremos calcular el valor $f(x) \in \mathbb{R}$. En el computador x es aproximado con un número racional $\tilde{x}$, así que $\tilde{x} - x$ es el *error inicial* y $\epsilon_1 = f(\tilde{x}) - f(x)$ es el correspondiente *error propagado.* En muchos casos, en vez de f se usa una función más simple f_1 (a menudo una expansión *truncada* de f). La diferencia $\epsilon_2 = f_1(\tilde{x}) - f(\tilde{x})$ es el *error de truncación.* Luego, las operaciones que hace el computador son "seudo-operaciones" (por el redondeo) por lo que en vez de $f_1(\tilde{x})$ se obtiene otro valor incorrecto $f_2(\tilde{x})$. La diferencia $\epsilon_3 = f_2(\tilde{x}) - f_1(\tilde{x})$ se podría llamar *el error propagado por los redondeos.* El error total sería

$$\epsilon = f_2(\tilde{x}) - f(x) = \epsilon_1 + \epsilon_2 + \epsilon_3$$

La aritmética usada por el computador no respeta la aritmética ordinaria. Cada simple operación en punto flotante casi siempre genera un error pequeño que se puede propagar en las siguientes operaciones. Se puede minimizar los errores de redondeo incrementando el número de cifras significativas en el computador y así el error no será en general muy dañino excepto en casos particulares, por ejemplo en los que se restan cantidades de signo opuesto y muy parecidas en valor absoluto.

Análisis del error de propagación. El análisis de error es importante cuando se quiere investigar y/o garantizar el desempeño de los métodos numéricos usados en problemas teóricos y prácticos. Ejemplos catastróficos producidos por errores de redondeo se puede ver en `http://mathworld.wolfram.com/RoundoffError.html`.

Sea $\rho = \left|\dfrac{x - \text{rd}(x)}{x}\right|$. Se acostumbra escribir

$$\text{rd}(x) = x(1+\varepsilon),\ \ |\varepsilon| = \rho,\ \ |\varepsilon| \le \text{eps}$$

Para seguir adelante necesitamos definir un *modelo* para la aritmética de la computadora. Excepto por la ocurrencia de sobreflujo o subflujo, vamos a suponer en nuestro modelo que las operaciones $+,-,\cdot,/$ producen un resultado redondeado que es *representable* en el computador. Denotamos con $fl(x+y), fl(x\cdot y),...$ el resultado de estas operaciones (el resultado que produce la máquina). Entonces, por ejemplo

$$fl(x\cdot y) = x\cdot y\,(1+\varepsilon),\ \ |\varepsilon| \le \text{eps}$$

Ahora bien, nuestro interés es analizar el error en los resultados causados por errores en los datos. Vamos a suponer que $x(1+\varepsilon_x)$ y $y(1+\varepsilon_y)$ son valores de x e y contaminados con errores relativos ε_x y ε_y. Analicemos el error relativo en cada operación $\cdot,+,-,/$.

- *Multiplicación.* Supongamos que ε_x y ε_y son tan pequeños que los términos de orden dos ε_x^2, ε_y^2, $\varepsilon_x\cdot\varepsilon_y,...$ u orden superior, puedan ser despreciados (respecto a los $\varepsilon's$). Entonces

$$x\,(1+\varepsilon_x)\cdot y(1+\varepsilon_y) \approx x\cdot y(1+\varepsilon_x+\varepsilon_y)$$

y por tanto, el error relativo $\varepsilon_{xy} \approx \varepsilon_x + \varepsilon_y$

Así, en general, en el producto los errores relativos de los datos se suman en el resultado. Esta situación la vamos a considerar *aceptable.*

- *División.* Si $y \neq 0$ entonces, usando la expansión en serie de $1/(x+1)$,

$$\frac{x\,(1+\varepsilon_x)}{y\,(1+\varepsilon_y)} = \frac{x}{y}\,(1+\varepsilon_x)(1-\varepsilon_y+\varepsilon_y^2-...) \approx \frac{x}{y}\,(1+\varepsilon_x-\varepsilon_y)$$

y entonces el error relativo $\varepsilon_{x/y} \approx \varepsilon_x - \varepsilon_y$ el cual es aceptable.

- *Suma y resta.* Como x, y pueden tener cualquier signo, basta con considerar la suma.

$$\begin{aligned} x\,(1+\varepsilon_x) + y\,(1+\varepsilon_y) &= x + y + x\,\varepsilon_x + y\,\varepsilon_y \\ &= (x+y)\left(1 + \frac{x\,\varepsilon_x + y\,\varepsilon_y}{x+y}\right) \end{aligned}$$

si $x + y \neq 0$. Entonces

$$\varepsilon_{x+y} = \frac{x}{x+y}\,\varepsilon_x + \frac{y}{x+y}\,\varepsilon_y$$

que es nuevamente una combinación lineal de los errores en los datos.

  - Si x e y son de *igual signo* entonces $0 \leq \dfrac{x}{x+y} \leq 1$, luego

$$\varepsilon_{x+y} \leq |\varepsilon_x| + |\varepsilon_y|$$

  que es un resultado aceptable.

  - Si x e y son de *signo contrario* entonces $\dfrac{x}{x+y}$ y $\dfrac{y}{x+y}$ pueden ser números arbitrariamente grandes cuando $|x+y|$ es arbitrariamente pequeño comparado con $|x|$ e $|y|$. Y esto ocurre cuando x e y son casi iguales en valor absoluto, pero de signo contrario. Este fenómeno es el llamado *error de cancelación.* Es un talón de Aquiles del análisis numérico y debe ser evitado siempre que sea posible. Observe que los efectos de la "cancelación" se pueden alcanzar por la suma de pequeñas dosis de cancelación en grandes cálculos.

Estabilidad y condición. En los cálculos científicos con el computador, los datos de entrada pueden ser imprecisos. Los errores en la entrada se propagan y dan lugar a errores en la salida. También los errores de redondeo en cada paso de un cálculo se propagan, provocando errores en el resultado final. Para muchos algoritmos, un análisis de error de redondeo se puede hacer para mostrar que pequeños cambios en los datos de entrada solo provocan pequeños cambios en le resultado (el algoritmo esta *bien condicionado*) y de paso se puede estimar el efecto de los errores de redondeo en el resultado final.

Aquí vamos a ver brevemente como se estudia la propagación de errores en el caso de operaciones aritméticas en el computador.

Supongamos que $\boldsymbol{x} = (x_1, x_2, ...x_n)$ son los datos de entrada y $\phi(\boldsymbol{x})$ representa el conjunto de operaciones (generalmente operaciones racionales). Sea ϕ es continuamente diferenciable en un conjunto abierto de $\mathbb{R}^n$ que contiene el dato $\boldsymbol{x}$. Para estudiar como los errores en $\boldsymbol{x}$ afectan el resultado $y = \phi(\boldsymbol{x})$ se usa una expansión (truncada) de Taylor,

$$\Delta y = \frac{\partial \phi}{\partial x_1}(\boldsymbol{x})\,\Delta x_1 + \frac{\partial \phi}{\partial x_2}(\boldsymbol{x})\,\Delta x_2 + ... + \frac{\partial \phi}{\partial x_n}(\boldsymbol{x})\,\Delta x_n$$

donde $\Delta x_1, \Delta x_2, ..., \Delta x_n$ son los errores individuales ($\Delta x_i = \tilde{x}_i - x_i$). Δy es una aproximación de primer orden del error $\phi(\tilde{\boldsymbol{x}}) - \phi(\boldsymbol{x})$ y la aproximación de primer orden para el error relativo sería $\dfrac{\Delta y}{y}$. Es conveniente escribir esta última expresión como

$$\frac{\Delta y}{y} = \frac{x_1}{\phi(\boldsymbol{x})}\frac{\partial \phi}{\partial x_1}(\boldsymbol{x})\,\frac{\Delta x_1}{x_1} + \frac{x_1}{\phi(\boldsymbol{x})}\frac{\partial \phi}{\partial x_2}(\boldsymbol{x})\,\frac{\Delta x_2}{x_2} + ... + \frac{x_1}{\phi(\boldsymbol{x})}\frac{\partial \phi}{\partial x_n}(\boldsymbol{x})\,\frac{\Delta x_n}{x_n}$$

Los números $\dfrac{x_i}{\phi(\boldsymbol{x})}\dfrac{\partial \phi}{\partial x_i}(\boldsymbol{x})$ se llaman *números de condición* del problema $y = \phi(x)$. Si esto números, en valor absoluto, son > 1 entonces *magnifican* el error y el cálculo podría volverse numéricamente *inestable* mientras que si estos números de condición, en valor absoluto, son ≤ 1 entonces el error no es magnificado.

- En el caso de sumas y restas, podemos calcular una cota para los errores absolutos en el resultado:

$$y = x_1 + x_2 + ... + x_n, \quad \text{entonces} \quad |\Delta y| \leq |\Delta x_1| + |\Delta x_2| + ... + |\Delta x_n|$$

  Si $y = x_1 \pm x_2$, los números de condición son $\left|\dfrac{x_i}{y(\boldsymbol{x})}\dfrac{\partial y}{\partial x_i}(\boldsymbol{x})\right| = \left|\dfrac{x_i}{x_1 + x_2}\right|$. Este número de condición es ≤ 1 si los todos los sumandos x_i tiene el mismo signo, en este caso el error no es magnificado. Pero si los sumandos tienen signos opuestos y son parecidos en valor absoluto entonces el error se magnifica.

- En el caso de productos o divisiones, si $y = x_1^{m_1} x_2^{m_2} \ldots x_n^{m_n}$ entonces

$$\left|\frac{\Delta y}{y}\right| \lessapprox \sum_{i=1}^{n} |m_i| \left|\frac{\Delta x_i}{x_i}\right|$$

  Si $y = x_1 x_2 \ldots x_n$, los números de condición son $\left|\dfrac{x_i}{y(\boldsymbol{x})}\dfrac{\partial y}{\partial x_i}(\boldsymbol{x})\right| = 1$, entonces el error no es magnificado.

- Si $y = f(x)$, como $\Delta y = f(x + \Delta x) - f(x) = f'(\xi)\Delta x$ con ξ entre x y $x + \Delta x$, entonces si $|\Delta x| \leq \epsilon$, tenemos

$$\Delta y \leq \max_{\xi} |f'(\xi)| \epsilon \quad \text{con} \quad \xi \in [x - \epsilon,\, x + \epsilon]$$

  En la práctica se usa sustituir ξ por una estimación (disponible) de x.

En general, si $y = f(x_1, ..., x_n)$ es diferenciable en un entorno abierto de $\boldsymbol{x}$, entonces

$$\Delta f \approx \sum_{i=1}^{n} \frac{\partial f}{\partial x_i} \Delta x_i \quad \text{y} \quad |\Delta f| \lessapprox \sum_{i=1}^{n} \left|\frac{\partial f}{\partial x_i}\right|_{\boldsymbol{x}} |\Delta x_i|$$

En esta fórmula se puede usar el máximo absoluto de las derivadas parciales, pero aún así, las cotas de error pueden salir sobreestimadas. Usando probabilidades se puede establecer algo más manejable: Si los errores Δx_i se ven como

variables aleatorias independientes con media cero y desviaciones estándar ϵ_1, ϵ_2, ...,ϵ_n, entonces el error estándar ϵ para $f(x_1,...,x_n)$ es

$$\epsilon \approx \sqrt{\sum_{i=1}^{n} \left(\frac{\partial f}{\partial x_i}\right)^2 \epsilon_i^2}$$

Las fórmulas anteriores nos dan un estimado del error en ciertos cálculos, pero en muchos casos la estimación del error más bien depende del cuidado y la experiencia que ayuda a evitar ciertos cálculos de riesgo.

EJERCICIOS

1.4 Consideremos el polinomio $P(x) = (x-2)^7(x-3)(x-4)$. En forma extendida, $P(x) = -1536 + 6272\,x - 11328\,x^2 + 11872\,x^3 - 7952\,x^4 + 3528\,x^5 - 1036\,x^6 + 194\,x^7 - 21\,x^8 + x^9$. Evaluar el polinomio, en sus dos formas, en $x = 1.99$

1.5 Supongamos que la ecuación cuadrática $ax^2 + bx + c = 0$ tiene dos soluciones distintas,

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

o alternativamente,

$$x_1^{'} = \frac{-2c}{b + \sqrt{b^2 - 4ac}}$$

$$x_2^{'} = \frac{-2c}{b - \sqrt{b^2 - 4ac}}$$

Si $b > 0$ evitamos la cancelación tomando x_2 y $x_1^{'}$.

Si $b < 0$ evitamos la cancelación tomando x_1 y $x_2^{'}$.

Usando los valores $a = 1,\ b = 1$ y $c = 10^{-k}, k = 1,2,3,...$, calcule las raíces de la cuadrática de dos maneras distintas, para investigar el error en la salida del computador debido al fenómeno de cancelación.

1.6 Considere la suma $S_1 = \displaystyle\sum_{n=1}^{2N} (-1)^n \frac{n}{n+1}$.

a) Muestre que $S_1 = -\displaystyle\sum_{n=1}^{N} \frac{2n-1}{2n} + \sum_{n=1}^{N} \frac{2n}{2n+1}$

b) Muestre que $S_1 = \displaystyle\sum_{n=1}^{N} \frac{1}{2n(2n+1)}$

c) Calcule la suma usando las dos expresiones para S_1 considerando valores grandes de N.

1.7 Considere las sumas $S_{\uparrow} = \displaystyle\sum_{n=1}^{N} \frac{1}{n}$ y $S_{\downarrow} = \displaystyle\sum_{n=N}^{1} \frac{1}{n}$. Considere valores grandes de N y calcule las dos sumas

1.8 Considere $f(x) = x - \text{sen}(x)$. Obtenga el polinomio de Taylor de orden 7, alrededor de cero, para esta función. Considere valores pequeños de x y calcule con las dos expresiones.

1.9 Resuelva el sistema $\begin{cases} 0.780x + 0.563y &= 0.217 \\ 0.457x + 0.330y &= 0.127 \end{cases}$ redondeando siempre a tres decimales. Comparar con la solución exacta $x = 1$, $y = 1$.



2 INTERPOLACIÓN POLINOMIAL. ASPECTOS PRÁCTICOS

2.1 Introducción

La interpolación polinomial es la base de muchos tipos de integración numérica y tiene otras aplicaciones teóricas. En la práctica a menudo tenemos una tabla de datos $\{(x_i, y_i),\ i = 0,1,2,...,n\}$, obtenida por muestreo o experimentación. Suponemos que los datos corresponden a los valores de una función f desconocida (a veces es conocida, pero queremos cambiarla por una función más sencilla de calcular). El "ajuste de curvas" trata el problema de construir una función que aproxime muy bien estos datos (es decir, a f). Un caso particular de ajuste de curvas es la interpolación polinomial: En este caso se construye un polinomio $P(x)$ que pase por los puntos de la tabla.

La interpolación polinomial consiste en estimar $f(x^*)$ con $P(x^*)$ si x^* no está en la tabla pero se puede ubicar entre estos valores. Una situación típica se muestra en el siguiente ejemplo en el que tenemos datos que relacionan temperatura con el segundo coeficiente virial.[2]
En el mundillo del ajuste de curvas hay varias alternativas,

- Usar un polinomio interpolante. Es el método de propósito general más usado.
- Usar trazadores (splines). Estas son funciones polinomiales a trozos.
- Usar Polinomios trigonométricos en $[0, 2\pi]$. Son la elección natural cuando la función f es periódica de periodo 2π.
- Usar sumas exponenciales. Se usan si conocemos que f presenta decaimiento exponencial conforme $x \longrightarrow \infty$.
- Si los datos son aproximados ("datos experimentales"), lo conveniente sería usar *Mínimos Cuadrados*

Aquí solo vamos a tratar con interpolación polinomial y trazadores cúbicos.

[2] El comportamiento de gases no ideales se describe a menudo con la *ecuación virial de estado*

$$\frac{PV}{RT} = 1 + \frac{B}{V} + \frac{C}{V^2} + ...,$$

donde P es la presión, V el volumen molar del gas, T es la temperatura Kelvin y R es la constante de gas ideal. Los coeficientes $B = B(T)$, $C = C(T)$,... son el segundo y tercer coeficiente virial, respectivamente. En la práctica se usa la serie truncada

$$\frac{PV}{RT} \approx 1 + \frac{B}{V}$$

Ejemplo 2.1

Considere los siguientes datos para el nitrógeno (N_2):

$T(K)$	100	200	300	400	450	500	600
$B(cm^3/mol)$	-160	-35	-4.2	9.0	?	16.9	21.3

donde T es la temperatura y B es el segundo coeficiente virial. ¿Cuál es el segundo coeficiente virial a $450K$?. Para responder la pregunta, usando interpolación polinomial, construimos un polinomio P que pase por los seis puntos de la tabla (ya veremos cómo), tal y como se muestra en la figura (2.1). Luego, el segundo coeficiente virial a $450K$ es aproximadamente $P(450) = 13.5cm^3/mol$.

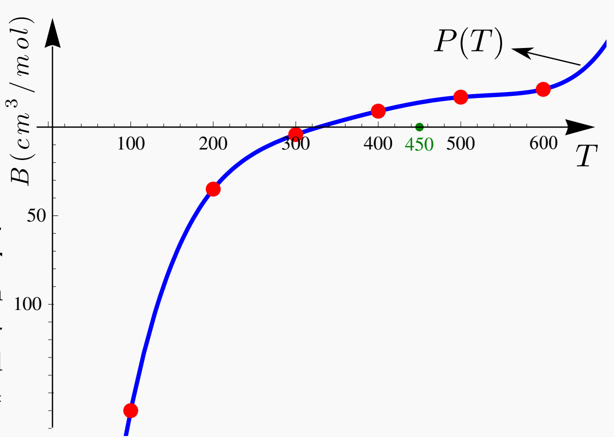


Figura 2.1 Polinomio interpolante

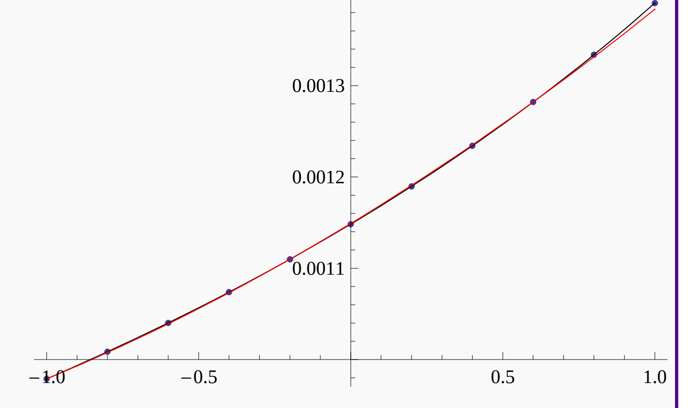
Ejemplo 2.2

Consideremos la función f definida por

$$f(x) = \int_5^\infty \frac{e^{-t}}{t-x}\,dt, \quad \text{con} \quad -1 \leq x \leq 1$$

La integral que define a f es una integral no trivial (no se puede expresar en términos de funciones elementales). La tabla de la izquierda nos muestra algunos valores para f.

x	$f(x)$
-1	0.0009788055864607286
-0.6	0.0010401386051341144
-0.2	0.0011097929435687336
0	0.0011482955912753257
0.2	0.0011896108201581322
0.25	**?**
0.6	0.0012820294923443982
1.	0.0013903460525251596



Podemos usar un polinomio interpolante para interpolar $f(0.25)$.

2.2 Interpolación polinomial.

Un problema de interpolación polinomial se especifica como sigue: dados $n+1$ *pares* $(x_0,y_0),(x_1,y_1),...,(x_n,y_n)$, siendo todos los x_i's distintos, y $y_i = f(x_i)$ para alguna función f; encontrar un polinomio $P_n(x)$ de grado $\leq n$ tal que

$$P_n(x_i) = y_i, \;\; i = 0,1,2,...,n \tag{2.1}$$

Teorema 2.1 (Polinomio interpolante).

Dados $n+1$ puntos (x_0,y_0), (x_1,y_1), ... , (x_n,y_n) con $x_i \neq x_j$ si $i \neq j$; existe un único polinomio $P_n(x)$ de grado $\leq n$ tal que $P(x_i) = y_i \;\; \forall\, i = 0,1,...,n$

A $P_n(x)$ se le llama *polinomio interpolante*, a cada x_i le decimos *nodo de interpolación* y a cada y_i *valor interpolado.*

- El problema tiene solución única, es decir hay un único polinomio que satisface (2.1).
- No se requiere que los datos estén igualmente espaciados ni en algún orden en particular.
- Si f es un polinomio de grado $k \leq n$, el polinomio interpolante de f en $n+1$ puntos coincide con f.
- El grado de P_n es $\leq n$ pues podría pasar, por ejemplo, que tres puntos estén sobre una recta y así el polinomio tendría grado cero o grado uno

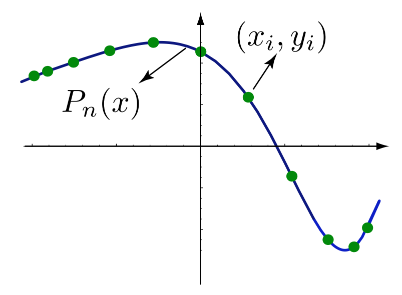


Figura 2.2 Polinomio interpolante.

Definición 2.1

Si de una función f conocemos los puntos (x_0,y_0), (x_1,y_1), ... , (x_n,y_n), con los x_i's todos distintos, y si $A = \{x_0,x_1,...,x_n\}$ y $x^* \notin A$ pero $\min A < x^* < \max A$; entonces *interpolar* f en x^* con un subconjunto de $k+1$ nodos de A consiste en calcular $P_k(x^*)$ donde P_k es el polinomio interpolante obtenido con un subconjunto de $k+1$ nodos alrededor de x^*.

El polinomio interpolante es único, es decir, solo hay un polinomio que pasa por estos $n+1$ puntos. Aquí vamos a ver cuatro maneras de calcular este polinomio interpolante: La forma de Lagrange del polinomio interpolante, la fórmula baricéntrica de Lagrange, la modificada de Lagrange y la forma de Newton del polinomio interpolante (método de diferencias divididas de Newton). Los cuatro métodos dan el mismo polinomio (aunque con diferente aspecto), y los cuatro métodos son importantes porque de ellos se hacen otras derivaciones teóricas.

2.3 Forma de Lagrange del polinomio interpolante.

Lagrange[3] calculó el único polinomio interpolante de manera explícita: El polinomio $P_n(x)$ de grado $\leq n$ que pasa por los $n+1$ puntos $(x_0,y_0),(x_1,y_1),...,(x_n,y_n)$ (con $x_i \neq x_j$ para todo i,j) es

$$P_n(x) = y_0 L_{n,0}(x) + y_1 L_{n,1}(x) + ... + y_n L_{n,n}(x)$$

donde $L_{n,k}(x) = \prod_{\substack{i=0 \\ i\neq k}}^{n} \frac{x-x_i}{x_k-x_i} = \frac{(x-x_0)(x-x_1)\cdots(x-x_{k-1})\curvearrowright(x-x_{k+1})\cdots(x-x_n)}{(x_k-x_0)\cdots(x_k-x_{k-1})\curvearrowright(x_k-x_{k+1})\cdots(x_k-x_n).}$

Por ejemplo,

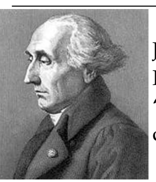
$$\begin{aligned}
L_{n,0}(x) &= \frac{(x-x_1)\cdot(x-x_2)\cdots(x-x_n)}{(x_0-x_1)\cdot(x_0-x_2)\cdots(x_0-x_n).} \\
L_{n,1}(x) &= \frac{(x-x_0)\cdot(x-x_2)\cdots(x-x_n)}{(x_1-x_0)\cdot(x_1-x_2)\cdots(x_1-x_n).} \\
L_{n,3}(x) &= \frac{(x-x_0)\cdot(x-x_2)\cdot(x-x_4)\cdots(x-x_n)}{(x_3-x_0)\cdot(x_3-x_2)\cdot(x_3-x_4)\cdots(x_3-x_n).} \\
\vdots \quad & \vdots \quad \vdots \\
L_{n,n}(x) &= \frac{(x-x_0)\cdot(x-x_1)\cdots(x-x_{n-1})}{(x_n-x_0)\cdot(x_n-x_1)\cdots(x_n-x_{n-1}).}
\end{aligned}$$

Ejemplo 2.3

Determine la forma de Lagrange polinomio interpolante de grado ≤ 2 (una recta o una parábola) que pasa por tres puntos $(0,1),(1,3),\ (2,0)$.

Solución:

$$\begin{aligned}
P_2(x) &= y_0\,L_{2,0}(x) + y_1\,L_{2,1}(x) + y_2\,L_{2,2}(x) \\
&= 1\cdot L_{2,0}(x) + 3\cdot L_{2,1}(x) + 0\cdot L_{2,2}(x) \\
&= 1\cdot\frac{(x-1)(x-2)}{(0-1)(0-2)} + 3\cdot\frac{(x-0)(x-2)}{(1-0)(1-2)}
\end{aligned}$$

[3]  Joseph Louis Lagrange (1736-1813) fue uno de los más grandes matemáticos de su tiempo. Nació en Italia pero se nacionalizó Francés. Hizo grandes contribuciones en todos los campos de la matemática y también en mecánica. Su obra principal es la "Mécanique analytique"(1788). En esta obra de cuatro volúmenes, se ofrece el tratamiento más completo de la mecánica clásica desde Newton y sirvió de base para el desarrollo de la física matemática en el siglo XIX.

Ejemplo 2.4

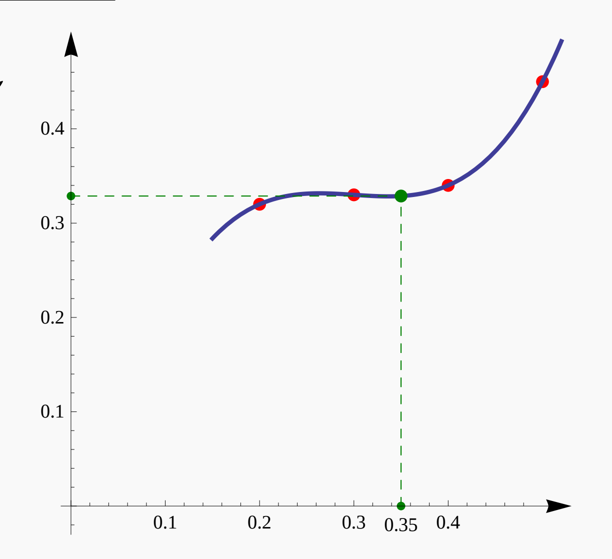
De una función f, conocemos la información de la tabla que sigue. Interpolar $f(0.35)$ usando un polinomio interpolante $P_3(x)$ indicando la subtabla de datos que va a usar.

x	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
$f(x)$	.3	.31	.32	.33	.34	.45	.46	.47

Solución: Como se requiere un polinomio interpolante $P_3(x)$, se necesita una subtabla de *cuatro* datos. Una opción es

x	0.2	0.3	0.4	0.5
$f(x)$	0.32	0.33	0.34	0.45

Si usamos la forma de Lagrange del polinomio interpolante, entonces



$$\begin{aligned}
P_3(x) &= 0.32 \cdot \frac{(x-0.3)(x-0.4)(x-0.5)}{(0.2-0.3)(0.2-.4)(0.2-0.5)} \\
&+ 0.33 \cdot \frac{(x-0.2)(x-0.4)(x-0.5)}{(0.3-0.2)(0.3-0.4)(0.3-0.5)} \\
&+ 0.34 \cdot \frac{(x-0.2)(x-0.3)(x-0.5)}{(0.4-0.2)(0.4-0.3)(0.4-0.5)} \\
&+ 0.45 \cdot \frac{(x-0.2)(x-0.3)(x-0.4)}{(0.5-0.2)(0.5-0.3)(0.5-0.4)}
\end{aligned}$$

y entonces $f(0.35) \approx P_3(0.35) = 0.32875$.

Ejemplo 2.5 (Interpolación lineal. Fórmula de un solo punto para una recta).

Verifique que el polinomio interpolante de grado ≤ 1 que pasa por $(x_0,y_0),(x_1,y_1)$ es,

$$P_1(x) = m(x-x_1) + y_1 \; = \; \frac{(y_0-y_1)}{(x_0-x_1)}\,(x-x_1) + y_1$$

Solución: Usando la fórmula de Lagrange,

$$\begin{aligned}
P_1(x) &= y_0 L_{n,0}(x) + y_1 L_{n,1}(x) \\
&= y_0 \frac{(x-x_1)}{(x_0-x_1)} + y_1 \frac{(x-x_0)}{(x_1-x_0)}. \quad \text{Simplificando,} \\
&= \frac{(y_0-y_1)}{(x_0-x_1)}\,(x-x_1) + y_1
\end{aligned}$$

Ejemplo 2.6

En la tabla que sigue aparece las estadísticas de un curso con la cantidad de estudiantes en cada rango de notas.

Rango de Notas	30-40	40-50	50-60	60-70	70-80
N^o Estudiantes	35	48	70	40	22

Estime la cantidad de estudiantes con nota menor o igual a 55.

Solución: Para hacer la estimación necesitamos una tabla con las frecuencias acumuladas,

x≤	40	50	60	70	80
y	35	83	153	193	215

Ahora calculamos el polinomio interpolante,

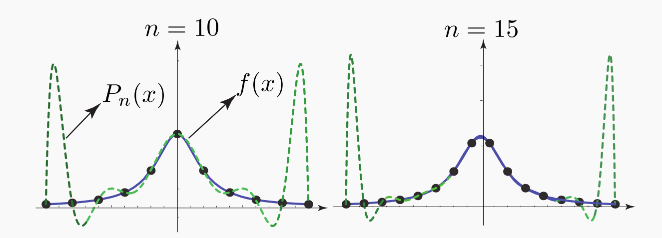
$$\begin{aligned} P_4(x) &= \frac{7(x-80)(x-70)(x-60)(x-50)}{48000} \\ &+ \frac{83(80-x)(x-70)(x-60)(x-40)}{60000} \\ &+ \frac{153(x-80)(x-70)(x-50)(x-40)}{40000} \\ &+ \frac{193(80-x)(x-60)(x-50)(x-40)}{60000} \\ &+ \frac{43(x-70)(x-60)(x-50)(x-40)}{48000} \end{aligned}$$

Así, la cantidad de estudiantes con nota menor o igual a 55 es aproximadamente $P_4(x) = 120$.

Ejemplo 2.7 (Nodos igualmente espaciados-fenómeno de Runge).

En general, el polinomio interpolante se podría ver afectado por el conjunto $\{x_0, ..., x_n\}$ y por la función f.

Este ejemplo es algo extremo y es conocido como 'fenómeno de Runge'; si $f(x) = \dfrac{1}{1+25x^2}$, el polinomio interpolante presenta problemas de convergencia si tomamos los x_i's igualmente espaciados en $[-1,1]$, es decir si $x_i = -1 + i \cdot h$ con $h = 2/n$.



Observe que la interpolación se ve afectado hacia los extremos del intervalo no asi en el centro; esto parece ser una tendencia general.

Si se puede escoger los nodos, una buena opción de ajuste se obtiene con nodos de Tchebychev [4]

Ejemplo 2.8 (Nodos de Tchebychev).

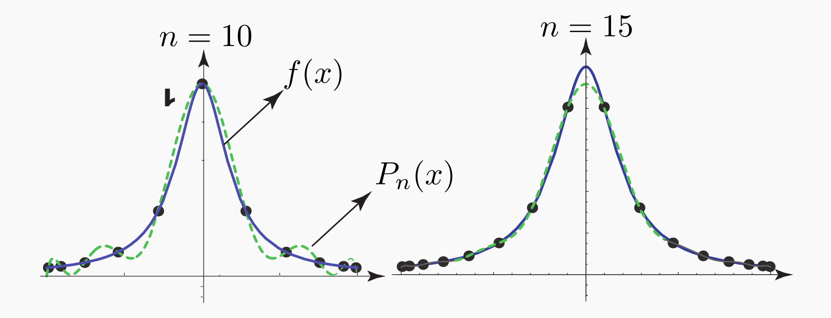
Si hay posibilidad de escoger los puntos de interpolación, en el intervalo $[-1,1]$, la elección podría ser los nodos

$$x_i = \cos\left(\frac{2i+1}{2n+2}\pi\right),$$

conocidos como nodos de Tchebychev. A diferencia de lo que podría suceder con nodos igualmente espaciados, con estos nodos el polinomio interpolante ajusta bien si $f \in C^1[-1,1]$.

Para un intervalo $[a,b]$ es válido hacer el cambio de variable $u = \dfrac{(b-a)(x-1)}{2} + b$ que mapea el intervalo $[-1,1]$ en el intervalo $[a,b]$. En este caso, los nodos serían

$$u_i = \frac{(b-a)(x_i-1)}{2+b} \quad \text{con} \quad x_i = \cos\left(\frac{2i+1}{2n+2}\pi\right).$$



Como se prueba más adelante, en este caso, si $x^* \in [a,b]$,

$$|f(x^*) - P_n(x^*)| \leq \frac{M}{(n+1)!}\frac{1}{2^n} \quad \text{si} \quad |f^{(n+1)}(x)| \leq M \quad \text{para todo} \quad x \in [a,b].$$

2.4 Forma modificada y forma baricéntrica de Lagrange.

La forma de Lagrange del polinomio interpolante es atractiva para propósitos teóricos. Sin embargo se puede reescribir en una forma que se vuelva eficiente para el cálculo computacional además de ser numéricamente mucho más estable (ver [2]). La forma modificada y la forma baricéntrica de Lagrange son útiles cuando queremos interpolar una función en todo un intervalo con un con un polinomio interpolante.

[4]  Pafnuti Lvóvich Tchebychev (1821 - 1894). El más prominente miembro de la escuela de matemáticas de St. Petersburg. Hizo investigaciones en Mecanismos, Teoría de la Aproximación de Funciones, Teoría de los Números, Teoría de Probabilidades y Teoría de Integración. Sin embargo escribió acerca de muchos otros temas: formas cuadráticas, construcción de mapas, cálculo geométrico de volúmenes, etc.

Supongamos que tenemos $n+1$ nodos distintos $x_0, x_1, ..., x_n$. Sea $\ell(x) = \prod_{i=0}^{n}(x - x_i)$ es decir,

$$\ell(x) = (x - x_0)(x - x_1)\cdots(x - x_n)$$

Definimos los *pesos baricéntricos* como

$$\omega_k = \prod_{\substack{i=0 \\ i\neq k}}^{n} \frac{1}{x_k - x_i}, \quad k = 0, 1, ..., n.$$

Es decir,

$$\omega_k = \frac{1}{x_k - x_0} \cdot \frac{1}{x_k - x_1} \cdots \frac{1}{x_k - x_{k-1}} \cdot \frac{1}{x_k - x_{k+1}} \cdots \frac{1}{x_k - x_n}.$$

Ahora podemos definir la "forma modificada" y "forma baricéntrica" de Lagrange:

Definición 2.2

La *forma modificada* del polinomio de Lagrange se escribe como

$$P_n(x) = \ell(x) \sum_{j=0}^{n} \frac{\omega_j}{x - x_j} y_j \tag{2.2}$$

Definición 2.3

La *forma baricéntrica* del polinomio de Lagrange se escribe

$$P_n(x) \begin{cases} = \; y_i & \text{si} \quad x = x_i, \\ = \; \dfrac{\displaystyle\sum_{k=0}^{n} \frac{\omega_k}{x - x_k} y_k}{\displaystyle\sum_{k=0}^{n} \frac{\omega_k}{x - x_k}}, & \text{si} \quad x \neq x_i \end{cases} \tag{2.3}$$

Ejemplo 2.9

Consideremos la siguiente tabla de datos,

x	$f(x)$
0.2	3.2
0.3	3.3
0.4	3.4
0.5	4.5

Calcule la forma modificada y la forma baricéntrica de Lagrange e interpole con ambos polinomios, $f(0.35)$.

Solución: Primero calculamos $\ell(x) = (x-0.2)(x-0.3)(x-0.4)(x-0.5)$. Ahora, los pesos baricéntricos,

$$\begin{aligned}
\omega_0 &= \frac{1}{0.2-0.3}\cdot\frac{1}{0.2-0.4}\cdot\frac{1}{0.2-0.5} &= -166.667,\\
\omega_1 &= \frac{1}{0.3-0.2}\cdot\frac{1}{0.3-0.4}\cdot\frac{1}{0.3-0.5} &= 500,\\
\omega_2 &= \frac{1}{0.4-0.2}\cdot\frac{1}{0.4-0.3}\cdot\frac{1}{0.4-0.5} &= -500,\\
\omega_3 &= \frac{1}{0.5-0.2}\cdot\frac{1}{0.5-0.3}\cdot\frac{1}{0.5-0.4} &= 166.667
\end{aligned}$$

Entonces, la forma modificada de Lagrange es,

$$P_3(x) = (x-0.2)(x-0.3)(x-0.4)(x-0.5)\left(-\frac{533.333}{x-0.2}+\frac{1650.}{x-0.3}-\frac{1700.}{x-0.4}+\frac{750.}{x-0.5}\right),$$

y la forma baricéntrica es,

$$P_3(x) = \frac{-\dfrac{533.333}{x-0.2}+\dfrac{1650.}{x-0.3}-\dfrac{1700.}{x-0.4}+\dfrac{750.}{x-0.5}}{-\dfrac{166.667}{x-0.2}+\dfrac{500.}{x-0.3}-\dfrac{500.}{x-0.4}+\dfrac{166.667}{x-0.5}}$$

En ambos casos, $f(0.35) \approx P_3(0.35) = 3.2875$.

2.5 Forma baricéntrica con nodos igualmente espaciados.

La forma baricéntrica toma una forma especialmente simple cuando los nodos son igualmente espaciados.

Sea $h > 0$ y $x_k = x_0 + k\cdot h;\; k = 0,1,2...,n,$ entonces

$$\omega_m^{-1} = \prod_{\substack{k=0\\k\neq m}}^{n}(x_m - x_k) = \prod_{\substack{k=0\\k\neq m}}^{n}(x_0 + m\cdot h - x_0 - k\cdot h) = (-1)^{n-m}h^n m!(n-m)!$$

Ahora, como la fórmula (2.3) no cambia si cambiamos ω_m por $\omega_m^* = c\omega_m$ con $c \neq 0$, entonces tomando $c = (-1)^n h^n n!$, los pesos modificados se convierten en coeficientes binomiales con signo alternado,

$$\omega_m^* = (-1)^m \binom{n}{m},\ m = 0,1,...,n$$

Finalmente, la forma baricéntrica para nodos igualmente espaciados no depende del peso h y sus coeficientes son enteros,

$$P_n(x) \begin{cases} = \quad y_i & \text{si} \quad x = x_i, \\ = \quad \dfrac{\displaystyle\sum_{m=0}^{n} (-1)^m \binom{n}{m} \frac{y_k}{x - x_m}}{\displaystyle\sum_{m=0}^{n} (-1)^m \binom{n}{m} \frac{1}{x - x_m}}, & \text{si} \quad x \neq x_i \end{cases} \tag{2.4}$$

EJERCICIOS

2.1 Considere los cuatro puntos $(0,1),(1,2),(3,0),(4,4)$.

a) Calcule el polinomio interpolante $P_3(x)$, en la forma de Lagrange.

b) Verifique que efectivamente $P_4(x_i) = y_i$, es decir, $P_3(0) = 1, etc.$

c) Interpolar $f(3.5)$.

2.2 Considere los cuatro puntos $(0,1),(1,2),(3,0),(4,4)$. en la forma de modificada y la forma baricéntrica de Lagrange.

a) Calcule el polinomio interpolante $P_3(x)$, en la forma de modificada.

b) Calcule el polinomio interpolante $P_3(x)$, en la forma de Baricéntrica.

c) Verifique que efectivamente $P_3(x_i) = y_i$, es decir, $P(0) = 1, etc.$

d) Interpolar $f(3.5)$.

2.3 Consideremos la siguiente tabla de datos,

x	$f(x)$
0.2	1.2
0.3	5.3
0.4	9.4
0.5	10.5

Calcule la forma modificada y la forma baricéntrica de Lagrange e interpole $f(0.35)$. **Ayuda:** Estas fórmulas permiten reutilizar los cálculos!

2.4 Usando la forma de Lagrange del polinomio interpolante verifique que si $P(x)$ pasa por $(x_0,y_0),(x_1,y_1)$ entonces $P(x) = \dfrac{(y_0 - y_1)}{(x_0 - x_1)}(x - x_1) + y_1$. **Ayuda:** En algún momento de la simplificación debe sumar y restar $y_1 x_1$.

2.5 Considere la función de Bessel $J_0(x) = \dfrac{1}{\pi}\displaystyle\int_0^{\pi} \cos(x \operatorname{sen}\theta)\, d\theta$. Tenemos la siguiente información,

x	$\pi J_0(x)$
0	3.59
0.2	3.11
0.4	3.08

a) Obtener la forma de Lagrange del polinomio interpolante.

b) Interpolar $J_0(0.25)$

2.6 Considere la siguiente tabla de salarios,

Salarios ($)	0-1000	1000-2000	2000-3000	3000-4000
Frecuencia	9	30	35	42

Estimar la cantidad de personas con salario entre $1000 y $1500.

2.7 Interpolar $\cos(1.75)$ usando la tabla

x_i	$\cos(1+3x_i)$
0	0.540302
1/6	0.070737
1/3	−0.416147

Ayuda: La estimación que se obtiene con el polinomio interpolante es -0.17054.

2.8 Considere la siguiente tabla de vapor para H_2O calentada a $200\,$MPa.

$v\ (m^3/kg)$	0.10377	0.11144	0.1254
$s\ (kJ/Kg\cdot K)$	6.4147	6.5453	6.7664

a) Use interpolación lineal para encontrar la entropía s para un volumen especifico v de $0.108m^3/kg$.
b) Use interpolación cuadrática para encontrar la entropía s para un volumen especifico v de $0.108m^3/kg$.

2.9 Usando la tabla del ejemplo (2.2), interpolar $f(0.25)$.

2.6 Forma de Newton para el polinomio interpolante.

La representación

$$P(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + \cdots + a_n(x-x_0)\cdots(x-x_{n-1}),$$

para el polinomio interpolante que pasa por los $n+1$ puntos $(x_0,y_0),...,(x_n,y_n)$, es conocida como *la representación de Newton* del polinomio interpolante.

2.7 Diferencias Divididas de Newton.

La manera más conocida para calcular la representación de Newton del polinomio interpolante, está basada en el método de *diferencias divididas.* Una gran ventaja sobre la forma clásica del método de Lagrange es que podemos agregar más nodos a la tabla de datos y obtener el polinomio interpolante sin tener que recalcular todo. Comparado con la forma modificada de Lagrange, no hay ganancia y más bien esta última forma es más estable. Aún así, el método de diferencias divididas tiene aplicaciones adicionales en otros contextos.

Podemos calcular los a_i's usando el hecho de que $P(x_i) = y_i$,

$$\begin{cases} P(x_0) = & y_0 & = & a_0 & \Longrightarrow & a_0 = y_0, \\ P(x_1) = & y_1 & = & a_0 + a_1(x_1-x_0) & \Longrightarrow & a_1 = \dfrac{y_1-y_0}{x_1-x_0} \\ P(x_2) = & y_2 & = & a_0 + a_1(x_2-x_0) + a_2(x_2-x_0)(x_2-x_1) & \Longrightarrow & a_2 = \dfrac{y_2-a_0-a_1(x_2-x_0)}{(x_2-x_0)(x_2-x_1)} \\ & & & \cdots & & \end{cases}$$

Si $y_k = f(x_k)$, la fórmula anterior nos muestra que cada a_k depende de $x_0, x_1, ..., x_k$. Desde muchos años atrás se usa la notación $a_k = f[x_0, x_1, ...x_k]$ para significar esta dependencia.

Al símbolo $f[x_0, x_1, ...x_n]$ se le llama *diferencia divida* de f. Usando esta nueva notación tendríamos que la forma de Newton del polinomio interpolante es

$$\begin{aligned} P(x) &= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) \\ &\quad + \cdots + f[x_0, ..., x_n](x - x_0)\cdots(x - x_{n-1}), \end{aligned}$$

donde $f[x_0] = y_0$ y $f[x_0, ..., x_i]$ es el coeficiente principal de la forma de Newton del polinomio que interpola la función f en los nodos $x_0, x_1, ..., x_i$.

Ejemplo 2.10 (Interpolación lineal).

El polinomio interpolante de grado ≤ 1 que pasa por $(x_0, y_0), (x_1, y_1)$ es

$$P_1(x) = f[x_0] + f[x_0, x_1](x - x_0) \quad \text{donde} \quad f[x_0, x_1] = \frac{(y_0 - y_1)}{(x_0 - x_1)} \quad \text{y} \quad f[x_0] = y_0$$

Si consideramos al coeficiente $f[x_0, x_1, ...x_n]$ como una función de $n+1$ variables, entonces esta función es *simétrica*, es decir, permutar las variables de cualquier manera no afecta el valor de la función. Esto es así porque el polinomio que interpola los puntos $\{(x_i, y_i)\}_{i=0,...,n}$ es único, por lo tanto sin importar el orden en que vengan los puntos, el coeficiente principal siempre es $a_n = f[x_0, x_1, ...x_n]$.

¿Qué es $f[x_k, x_{k+1}, ..., x_{k+j}]$?. Es el coeficiente principal de la forma de Newton del polinomio que interpola una función f en los nodos $x_k, x_{k+1}, ..., x_{k+j}$. Por ejemplo, si tenemos $n+1$ datos $(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)$, el polinomio que interpola $(x_3, y_3), (x_4, y_4)$ sería

$$P_1(x) = y_3 + f[x_3, x_4](x - x_3).$$

El nombre "*diferencia divida*" viene del hecho de que cada $f[x_k, x_{k+1}, ..., x_{k+j}]$ se puede expresar como un cociente de diferencias.

Teorema 2.2

La diferencia dividida $f[x_k, x_{k+1}, ..., x_{k+j}]$ satisface la ecuación

$$f[x_k, x_{k+1}, ..., x_{k+j}] = \frac{f[x_{k+1}, x_{k+2}, ...x_{k+j}] - f[x_k, x_{k+1}, ...x_{k+j-1}]}{x_{k+j} - x_k} \tag{2.5}$$

Ejemplo 2.11

El teorema (2.2) indica que cada diferencia dividida se puede calcular en términos de otras "diferencias" previamente calculadas. Los ejemplos que siguen son casos particulares para mostrar cómo se aplica el teorema.

$$\begin{aligned} f[x_i, x_j] &= \frac{y_i - y_j}{x_i - x_j} \\ f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} \\ f[x_1, x_2, x_3] &= \frac{f[x_2, x_3] - f[x_1, x_2]}{x_3 - x_1} \end{aligned}$$

Ejemplo 2.11 (continuación).

$$\begin{aligned} &\vdots \\ f[x_0, x_1, x_2, x_3] &= \frac{f[x_1, x_2, x_3] - f[x_0, x_1, x_2]}{x_3 - x_0} \\ f[x_1, x_2, x_3, x_4] &= \frac{f[x_2, x_3, x_4] - f[x_1, x_2, x_3]}{x_4 - x_1} \\ &\vdots \\ f[x_0, x_1, ..., x_k] &= \frac{f[x_1, x_2, ..., x_k] - f[x_0, x_1, ..., x_{k-1}]}{x_k - x_0} \end{aligned}$$

Este esquema recursivo se puede arreglar en forma matricial como sigue,

$$\begin{array}{llllll} x_0 & y_0 & & & \\ x_1 & y_1 & f[x_0, x_1] & & \\ x_2 & y_2 & f[x_1, x_2] & f[x_0, x_1, x_2] & \\ x_3 & y_3 & f[x_2, x_3] & f[x_1, x_2, x_3] & f[x_0, x_1, x_2, x_3] \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{array}$$

En general, para calcular $f[x_0]$, $f[x_0, x_1]$, $f[x_0, x_1, x_2], \cdots, f[x_0, ..., x_n]$, debemos calcular una matriz en la que las nuevas columnas se construyen con los datos de la columna anterior.

y_0, y_1, $f[x_0, x_1]$, y_2, $f[x_1, x_2]$, $f[x_0, x_1, x_2]$

$$f[x_0, x_1, x_2] = \frac{f[x_0, x_1] - f[x_1, x_2]}{x_2 - x_0}$$

La misma matriz se puede usar para calcular la forma de Newton para subconjuntos de datos: En el arreglo que sigue, la diagonal principal (en rojo) corresponde a los coeficientes del polinomio que interpola los datos $(x_0, y_{)}), ..., (x_n, y_n)$.

La diagonal en azul corresponde a los coeficientes del polinomio que interpola los datos $(x_1, y_1), ..., (x_n, y_n)$.

$$\begin{array}{llllll}
y_0 & & & & & \\
y_1 & f[x_0,x_1] & & & & \\
y_2 & f[x_1,x_2] & f[x_0,x_1,x_2] & & & \\
y_3 & f[x_2,x_3] & f[x_1,x_2,x_3] & & & \\
\vdots & \vdots & \vdots & & \ddots & \\
y_n & f[x_{n-1},x_n] & f[x_{n-2},x_{n-1},x_n] & \cdots & f[x_1,...,x_n] & f[x_0,x_1,...,x_n]
\end{array}$$

Por ejemplo, para calcular el polinomio que interpola los datos $(x_3, y_3), ..., (x_6, y_6)$ se usa la (sub)matriz,

$$\begin{array}{llll}
y_3 & & & \\
y_4 & f[x_3,x_4] & & \\
y_5 & f[x_4,x_5] & f[x_3,x_4,x_5] & \\
y_6 & f[x_5,x_6] & f[x_1,x_2,x_3] & f[x_3,x_4,x_5,x_6]
\end{array}$$

La diagonal principal (en rojo) corresponde a los coeficientes del polinomio que interpola estos cuatro datos.




http://www.tec-digital.itcr.ac.cr/revistamatematica/cursos-linea/NumericoApplets/DifDivNewton.htm

Ejemplo 2.12

Usando diferencias divididas, calcular el polinomio interpolante para los datos $(-1,2), (1,1), (2,2),$ $(3,-2)$ y el polinomio interpolante para los datos $(1,1), (2,2), (3,-2)$.

Solución: Primero construimos la matriz de diferencias divididas usando todos los datos. En rojo están los coeficientes del polinomio que interpola todos los datos y en azul los coeficientes del polinomio que interpola los datos $(1,1), (2,2), (3,-2)$.

$$\begin{array}{llllll}
x_0 & y_0 & & & \\
x_1 & y_1 & f[x_0,x_1] & & \\
x_2 & y_2 & f[x_1,x_2] & f[x_0,x_1,x_2] & \\
x_3 & y_3 & f[x_2,x_3] & f[x_1,x_2,x_3] & f[x_0,x_1,x_2,x_3]
\end{array}
\longmapsto
\begin{array}{rrrr}
2 & & & \\
1 & -1/2 & & \\
2 & 1 & 1/2 & \\
-2 & -4 & -5/2 & -3/4
\end{array}$$

El polinomio interpolante, en la forma de Newton, para todos los datos es

$$P(x) = 2 - 1/2(x+1) + 1/2(x+1)(x-1) - 3/4(x+1)(x-1)(x-2)$$

El polinomio interpolante, en la forma de Newton, para los datos $(1,1), (2,2), (3,-2)$ es

$$P(x) = 1 + 1 \cdot (x-1) + -5/2(x-1)(x-2)$$

Ejemplo 2.13

De una función f, conocemos la información de la tabla (2.1). Interpolar $f(0.35)$ usando un polinomio interpolante $P_3(x)$. Primero que todo, escriba la tabla de datos que va a usar.

x	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
$f(x)$	3	3.1	3.2	3.3	3.4	4.5	4.6	4.7

Tabla 2.1

Solución: Como se requiere un polinomio interpolante $P_3(x)$, se necesita una tabla de cuatro datos. Una opción es

x	0.2	0.3	0.4	0.5
$f(x)$	3.2	3.3	3.4	4.5

Si usamos la forma de Newton del polinomio interpolante, entonces

3.2			
3.3	1		
3.4	1	0	
4.5	11	50	$166.6\overline{6}$

$$\begin{aligned} P_3(x) &= 3.2 + 1\cdot(x-0.2) \\ &+ 0\cdot(x-0.2)(x-0.3) \\ &+ 166.6\overline{6}\cdot(x-0.2)(x-0.3)(x-0.4) \end{aligned}$$

Por tanto $f(0.35) \approx P_3(0.35) = 3.2875$

2.8 Forma de Newton en el caso de nodos igualmente espaciados.

Si tenemos nodos igualmente espaciados con $x_k = x_0 + k\cdot h, k = 0,1,,...,n,$ entonces la *diferencia hacia adelante* de orden 1 en y_k es $\Delta^1 y_k = y_{k+1} - y_k$. La diferencia hacia adelante de orden m se define recursivamente como: $\Delta^m y_k = \Delta(\Delta^{m-1} y_k)$. Así,

$$\begin{aligned} \Delta^0 y_k &:= y_k, \\ \Delta^1 y_k &= y_{k+1} - y_k, \\ \Delta^2 y_k &= \Delta(y_{k+1}-y_k) = y_{k+2} - y_{k+1} - y_{k+1} + y_k = y_{k+2} - 2y_{k+1} + y_k, \\ &\ldots \\ \Delta^n y_k &= \sum_{j=0}^{n} (-1)^j \binom{n}{j} y_{k+n-j} \end{aligned}$$

En particular

$$\Delta^n y_0 = \sum_{j=0}^{n} (-1)^j \binom{n}{j} y_{n-j}.$$

Recordemos que si $s \in \mathbb{R}$,

$$\begin{aligned}
\binom{s}{0} &= 1, \\
\binom{s}{1} &= s, \\
\binom{s}{2} &= \frac{s(s-1)}{2}, \\
\binom{s}{3} &= \frac{s(s-1)(s-2)}{6}, \\
\binom{s}{4} &= \frac{s(s-1)(s-2)(s-3)}{24}, \ldots
\end{aligned}$$

La relación entre estas diferencias hacia adelante y los coeficientes de la forma de Newton del polinomio interpolante (en el caso de nodos igualmente espaciados) se expresa mediante la fórmula,

$$k!\,h^k f[x_0, x_1, ..., x_k] = \Delta^k y_0.$$

De esta manera, la forma de Newton del polinomio interpolante, para nodos igualmente espaciados, es

$$P_n(x) = y_0 + (x-x_0)\frac{\Delta f(x_0)}{1!h} + (x-x_0)(x-x_1)\frac{\Delta^2 f(x_0)}{2!h^2} + \cdots + (x-x_0)(x-x_1)\cdots(x-x_{n-1})\frac{\Delta^n f(x_0)}{n!h^n}.$$

Se puede hacer una simplificación más; si $x = x_0 + s \cdot h$ entonces

$$\begin{aligned}
\frac{x-x_0}{h} &= s \\
\frac{x-x_i}{h} &= \frac{x-(x_0+i\cdot h)}{h} = s-i
\end{aligned}$$

De este modo,

$$\begin{aligned}
(x-x_0)\frac{\Delta f(x_0)}{1!h} &= \frac{(x-x_0)}{h}\Delta^1 f(x_0) = \binom{s}{1}\Delta^1 f(x_0), \\
(x-x_0)(x-x_1)\frac{\Delta^2 f(x_0)}{2!h^2} &= \frac{(x-x_0)(x-x_1)}{2!h^2}\Delta^2 f(x_0) = \frac{s(s-1)}{2!}\Delta^2 f(x_0) = \binom{s}{2}\Delta^2 f(x_0), \\
&\cdots \\
(x-x_0)(x-x_1)\cdots(x-x_{n-1})\frac{\Delta^n f(x_0)}{n!h^n} &= \frac{(x-x_0)(x-x_1)\cdots(x-x_{n-1})}{n!h^n}\Delta^n f(x_0) \\
&= \frac{s(s-1)(s-2)\ldots(s-n+1)}{n!}\Delta^n f(x_0) \\
&= \binom{s}{n}\Delta^n f(x_0).
\end{aligned}$$

Es decir, si los nodos son igualmente espaciados (de paso h) y $x = x_0 + s \cdot h$,

$$P_n(x) = P_n(x_0 + s\cdot h) = \sum_{k=0}^{n}\binom{s}{k}\Delta^k y_0 \; = \sum_{k=0}^{n}\binom{s}{k}k!\,h^k f[x_0, x_1, ... x_k].$$

Ejemplo 2.14

Usando la tabla de datos, interpolar $f(0.35)$.

x	$f(x)$
0.2	3.2
0.3	3.3
0.4	3.4
0.5	4.5

Solución: Los nodos son igualmente espaciados con $h = 0.1$. La matriz de diferencias divididas es,

$$\begin{array}{llll} 3.2 & & & \\ 3.3 & 1 & & \\ 3.4 & 1 & 0 & \\ 4.5 & 11 & 50 & 166.6\overline{6} \end{array}$$

Como $0.35 = 0.2 + 1.5 \cdot 0.1, \Longrightarrow s = 1.5$,

$$\begin{aligned} f(0.35) \approx P_3(0.35) &= \binom{1.5}{0} 0!\,(0.1)^0 \cdot 3.2 + \binom{1.5}{1} 1!\,(0.1)^1 \cdot 1 + \binom{1.5}{3} 3!\,(0.1)^3 \cdot 166.6\overline{6} \\ &= 1 \cdot 3.2 + 1.5 \cdot 0.1 \cdot 1 - 0.0625 \cdot 6 \cdot (0.1)^3 \cdot 166.6\overline{6} = 3.2875. \end{aligned}$$

EJERCICIOS

2.10 Sea $P(x) = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x^1 + a_0$, si se conoce que este polinomio pasa por $(-1,3), (0,0), (1,4), (2,0), (3,1), (4,0)$, determine los coeficientes "a_i" del polinomio.

2.11 Considere los datos $(x_0,1), (x_2,2), (x_3,3), (x_4,4), (x_5,5)$, donde $x_0 = 0.1, x_1 = 0.2, x_3 = 0.3, x_4 = 0.4$ y $x_5 = 0.5$. Calcule $f[x_2, x_3, x_4]$.

2.12 Verifique que $f[x_0,x_1,x_2] = \dfrac{y_0}{(x_0-x_1)(x_0-x_2)} + \dfrac{y_1}{(x_1-x_0)(x_1-x_2)} + \dfrac{y_2}{(x_2-x_0)(x_2-x_1)}$.

2.13 Considere los 4 datos $(0,1), (1,2), (3,0), (4,4)$.

a) Determine la matriz de diferencias divididas y la forma de Newton del polinomio interpolante $P_3(x)$.

b) Verifique que efectivamente $P_3(x_i) = y_i$, es decir, $P(0) = 1, etc.$

c) Interpolar $f(3.5)$.

2.14 Considere la siguiente tabla de datos para el nitrógeno,

$T(K)$	100	200	300	400	500	600
$B(cm^3/mol)$	-160	-35	-4.2	9.0	16.9	21.3

Tabla 2.2 Segundos Coeficientes viriales $B(cm^3/mol)$ para el nitrógeno

donde T es la temperatura y B es el segundo coeficiente virial. Interpolar el segundo coeficiente virial a $450K$.

2.15 Usar la forma de Newton del polinomio interpolante para completar la siguiente tabla de datos para el agua, donde T es temperatura y ρ es la densidad.

$T(C)$	50	60	65	68	75	80
$\rho(kg/m^3)$	988	985.7	980.5	?	974.8	971.6

Tabla 2.3

2.16 Verifique que $f[x_i, x_j] = f[x_j, x_i]$.

2.17 Usando la forma de Newton del polinomio interpolante, obtenga el polinomio $P_1(x)$ que pasa por $(x_6, y_6), (x_7, y_7)$.

2.18 Considere la función de Bessel $J_0(x) = \dfrac{1}{\pi}\displaystyle\int_0^{\pi} \cos(x \operatorname{sen}\theta)\, d\theta$. Tenemos la siguiente información,

x	$\pi J_0(x)$
0	3.59
0.2	3.11
0.4	3.08

a) Obtener la forma de Newton del polinomio interpolante.

b) Interpolar $J_0(0.25)$

2.19 En la tabla que sigue aparece las estadísticas de un curso con la cantidad de estudiantes en cada rango de notas.

Rango de Notas	30-40	40-50	50-60	60-70	70-80
N^o Estudiantes	35	48	70	40	22

a) Estime la cantidad de estudiantes con nota mayor o igual a 65.

b) Estime la cantidad de estudiantes en el rango $55 - 65$

2.20 La siguiente tabla muestra los pesos normales de bebés durante los primeros 12 meses de vida,

Edad en meses	0	2	5	8	10	12
Peso en libras	7.5	10.25	15	16	18	21

Determine el peso de los bebés entre los 5 y 5.6 meses de vida.

2.21 Interpolar $\cos(1.75)$ usando la tabla

x_i	$\cos(1+3x_i)$
0	0.540302
1/6	0.070737
1/3	-0.416147

Ayuda: la estimación que se obtiene con el polinomio interpolante es -0.17054.

2.22 Considere la siguiente tabla de vapor para H_2O calentada a 200 MPa.

$v\ (m^3/kg)$	0.10377	0.11144	0.1254
$s\ (kJ/Kg \cdot K)$	6.4147	6.5453	6.7664

a) Use interpolación lineal para encontrar la entropía s para un volumen especifico v de $0.108 m^3/kg$.

b) Use interpolación cuadrática para encontrar la entropía s para un volumen especifico v de $0.108m^3/kg$.

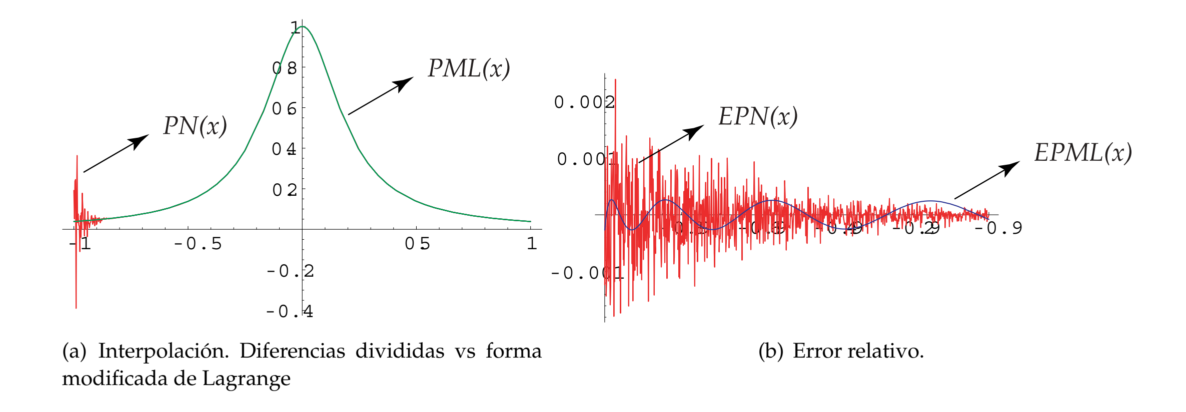
2.23 En la siguiente tabla de diferencias divididas, complete los datos que faltan.

x_i	y_i				
0	2				
1	3	□			
□	2	-1	-1		
3	1	□	0	□	
4	3	2	1.5	0.5	□

2.9 Forma de Lagrange vs Forma de Newton.

Usualmente se reserva la forma de Lagrange del polinomio interpolante para trabajo teórico y diferencias divididas de Newton para cálculos. La realidad es que la *forma modificada de Lagrange* es tan eficiente como diferencias divididas de Newton en cuanto a costo computacional y además es numéricamente mucho más estable. Hay varias ventajas que hacen de esta forma modificada de Lagrange, el método a escoger cuando de interpolación polinomial se trata ([8], [9]).

Para mostrar la inestabilidad del polinomio interpolante obtenido con diferencias divididas versus el obtenido con la forma modificada de Lagrange, consideramos la función de Runge $f(x) = 1/(1+25x^2)$ en $[-1,1]$. Para un buen ajuste, usamos 52 nodos de TChebyshev. En la figura (2.9,(a)) se muestra la gráfica de f junto con la gráfica del polinomio interpolante obtenido con diferencias divididas ($PN(x)$) y del polinomio interpolantes obtenido con la forma modificada de Lagrange ($PML(x)$). Usando la aritmética usual de la máquina, se nota inestabilidad de $PN(x)$ en las cercanías de $x = -1$. En la figura (2.9,(b)) se muestra el error relativo de la aproximación a f con cada polinomio en $[-1,-0.9]$. $EPN(x)$ corresponde al error relativo entre f y la forma de Newton del polinomio interpolante y $EPML(x)$ corresponde al error relativo entre f y la forma modificada de Lagrange.



(a) Interpolación. Diferencias divididas vs forma modificada de Lagrange

(b) Error relativo.

2.10 Estimación del error.

La estimación del error, cuando interpolamos con un polinomio interpolante, es de interés práctico en varias áreas, por ejemplo en el desarrollo de métodos de aproximación en ecuaciones diferenciales ordinarias y en ecuaciones

diferenciales en derivadas parciales.

Una estimación del error se puede obtener si conocemos alguna información acerca de la función f y sus derivadas. Sea $f \in C^{n+1}[a,b]$ y $P_n(x)$ el polinomio de interpolación de f en (x_0,y_0), (x_1,y_1), ..., (x_n,y_n), con $x_i \in [a,b]$. Entonces, usando polinomios de Taylor podemos establecer la siguiente fórmula para el error

$$f(x) - P_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x-x_0)(x-x_1)\cdots(x-x_n)$$

donde $a < \xi(x) < b$ y $x \in [a,b]$. Aquí, la expresión "$\xi(x)$" significa que ξ no es una constante fija, sino que varía según el valor que tome x.

Para efectos prácticos, a y b son el mínimo y el máximo del conjunto $\{x_0, x_1, ..., x_n\}$. Si M_n es el el máximo absoluto de la función $|f^{(n+1)}|$ en $[a,b]$, es decir, $|f^{(n+1)}(x)| \leq M_n$ para todo $x \in [a,b]$, entonces podemos obtener una estimación del error $f(x) - P_n(x)$ con la desigualdad,

$$|f(x) - P_n(x)| \leq \frac{M_n}{(n+1)!}|(x-x_0)(x-x_1)\cdots(x-x_n)|; \quad x \in [a,b]. \tag{2.6}$$

Observe que un polinomio interpolante de grado alto no garantiza una mejora en el error: Si usamos más puntos (posiblemente más cercanos entre ellos) se puede esperar que el producto $\prod_i (x - x_i)$ se haga más pequeño con n, pero todavía debería pasar que la derivada de orden $n+1$ no crezca más rápido que $(n+1)!$ y esto parece no ser la regla[5].

Si los nodos son igualmente espaciados, y suponiendo que tenemos n y M_n fijos, la estimación del error depende de la función $\ell(x) = (x-x_0)(x-x_1)\cdots(x-x_n)$. La forma general de esta función se muestra en la figura que sigue, Esto sugiere que en el caso de nodos igualmente espaciados (excepto $n = 1$), el error es más pequeño si x

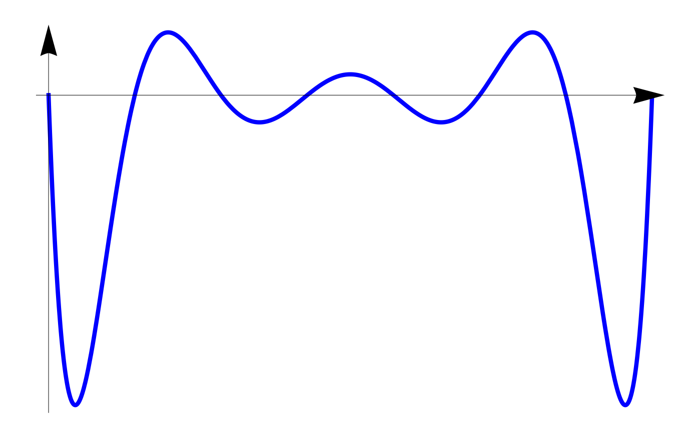


Figura 2.3 $\ell(x) = (x-x_0)\cdots(x-x_7)$ con 7 nodos igualmente espaciados.

está hacia el centro y empeora en los extremos.

[5] Georg Faber (1912) demostró que para cada juego de nodos, existe un función continua para la cual los polinomios interpolantes no convergen uniformemente a f y también, para cada función continua existe un juego de nodos donde los polinomios interpolantes si convergen de manera uniforme. Aún en este último caso, los nodos no siempre fáciles de obtener.

La desigualdad (2.6) sería suficiente para estimar el error al interpolar en un valor x, pero nos interesa también una estimación que nos sirva para todo $x \in [x_0, x_n]$.

2.11 Error en interpolación lineal.

Si tenemos dos puntos $(x_0, y_0), (x_1, y_1)$ con $x_0 < x_1$, el error es $f(x) - P_1(x) = \dfrac{(x-x_0)(x-x_1)}{2} f''(\xi(x))$. ¿Cuál es el error máximo si x está entre x_0 y x_1 y si f'' permanece acotada en $[x_0, x_1]$?.

Si $|f''(x)| \leq M_2$ en $[x_0, x_1]$, entonces

$$|f(x) - P_1(x)| \leq \frac{M_2}{2!} \left|(x-x_0)(x-x_1)\right|.$$

El error máximo depende del máximo valor de la función $\left|\dfrac{(x-x_0)(x-x_1)}{2}\right|$ en el intervalo $[x_0, x_1]$.
Como $\ell(x) = \dfrac{(x-x_0)(x-x_1)}{2}$ es una parábola cóncava hacia arriba (figura 2.11), es negativa si $x \in [x_0, x_1]$, por lo tanto el máximo en valor absoluto lo alcanza en $x = \dfrac{x_0+x_1}{2}$, y es

$$\frac{(x_1-x_0)^2}{8}.$$

∴ Si se usa interpolación lineal, el error general esta acotado por

$$|f(x) - P_1(x)| \leq M_2 \frac{(x_1-x_0)^2}{8}.$$

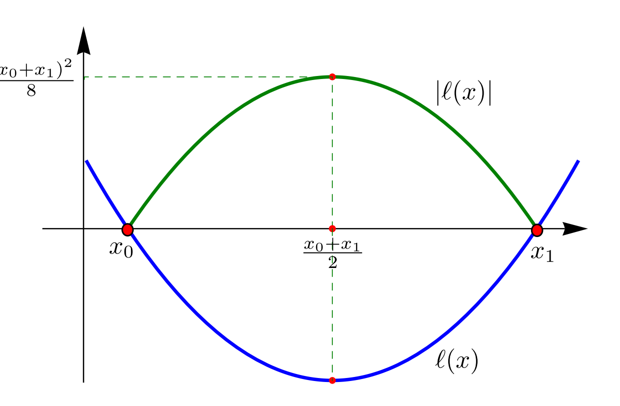


Figura 2.4 $\ell(x) = (x-x_0)(x-x_1)$ y $|\ell(x)|$

Ejemplo 2.15

Si tabulamos la función $\operatorname{sen} x$ para $x_0 = 0$, $x_1 = 0.002$, $x_2 = 0.004, \ldots$ entonces el error general al *interpolar linealmente* es

$$|\operatorname{sen} x - P_1(x)| \leq 1 \cdot \left|\frac{(0.002)^2}{8}\right| = 0.5 \times 10^{-6},$$

pues $|\operatorname{sen} x| \leq 1 \ \forall\, x$ (Aquí suponemos que el polinomio se evalúa de manera exacta). Esto nos dice que la función $\operatorname{sen} x$ es apropiada para interpolación lineal.

Si deseamos más precisión en un caso particular, podemos usar la fórmula (2.6). Si por ejemplo $x = 0.003$, entonces $|\operatorname{sen}(0.003) - P_1(0.003)| \leq \dfrac{\operatorname{sen}(0.004)}{2} |(0.003 - 0.002)(0.003 - 0.004)| \approx -1.99 \times 10^{-9}$ pues el máximo absoluto de $|\operatorname{sen} x|$ en el intervalo $[0.002, 0.004]$ es $\operatorname{sen}(0.004)$.

2.12 Error en interpolación cuadrática

Si interpolamos con tres puntos ($n = 2$) igualmente espaciados x_0, $x_1 = x + h$ y $x_2 = x_0 + 2h$; entonces si $x \in [x_0, x_2]$ y si $|f'''(x)| \leq M_3$ en $[a,b]$, la estimación general del error es,

$$\begin{aligned} |f(x) - P_2(x)| &\leq \frac{M_3}{3!}|(x-x_0)(x-x_1)(x-x_2)| \\ &\leq \frac{M_3}{6}|(x-x_0)(x-x_0-h)(x-x_0-2h)| \end{aligned}$$

Para obtener el máximo absoluto de la función $\ell(x) = (x-x_0)(x-x_1)(x-x_2)$ calculamos sus puntos críticos: $\ell'(x) = 3x^2 + x(-6h - 6x_0) + 6hx_0 + 3x_0^2 + 2h^2$, los ceros de esta cuadrática son

$$r_1 = \frac{1}{3}(3h + \sqrt{3}h + 3x_0) \ \text{ y } \ r_2 = \frac{1}{3}(3h - \sqrt{3}h + 3x_0).$$

Como $\ell(x)$ se anula en x_0 y x_2, el máximo absoluto de $|\ell(x)|$ es $\text{máx}\{|\ell(r_1)|, |\ell(r_2)|\} = \dfrac{2h^3}{3\sqrt{3}}$.

$\therefore$ El error general al interpolar con tres puntos igualmente espaciados es $|f(x) - P_2(x)| \leq \dfrac{M_3 h^3}{9\sqrt{3}}$, $x \in [x_0, x_2]$.

Ejemplo 2.16

Si tabulamos la función $\operatorname{sen} x$ para $x_0 = 0$, $x_1 = 0.01$, $x_2 = 0.02, \ldots$ entonces el error general al *interpolar con un polinomio de grado dos* es

$$|\operatorname{sen} x - P_2(x)| \leq \frac{1 \cdot (0.01)^3}{9\sqrt{3}} \approx 6.415 \times 10^{-8},$$

pues $|\operatorname{sen} x| \leq 1 \ \forall x$.

2.13 Error en interpolación cúbica

Si tenemos cuatro puntos *igualmente espaciados* $(x_0, y_0), (x_1, y_1), (x_2, y_2), (x_3, y_3)$ con $x_0 < x_1 < x_2 < x_3$, una estimación del error es

$$|f(x) - P_3(x)| \leq \frac{M_4}{4!}|(x-x_0)(x-x_1)(x-x_2)(x-x_3)|, \quad \text{con} \quad |f^{(4)}(x)| \leq M_4 \ \text{ en } \ [x_0, x_3].$$

De nuevo, dados n y M_4 fijos, la estimación del error general depende del máximo absoluto del polinomio $|\ell(x)| = |(x-x_0)(x-x_1)(x-x_2)(x-x_3)|$. Como $x_i = x_0 + i \cdot h, i = 1,2,3$;

$$\begin{aligned}\ell(x) &= (x-x_0)(x-x_1)(x-x_2)(x-x_3)\\ &= (x-x_0)(x-x_0-h)(x-x_0-2h)(x-x_0-3h)\end{aligned}$$

$$\ell'(x) = 2(2x-3h-2x_0)(x^2+x(-3h-2x_0)+h^2+3hx_0+x_0^2)$$

Los puntos críticos son $r_1 = 0.5(3h+2x_0)$, $r_2 = 0.5(3h-\sqrt{5}h+2x_0)$ y $r_3 = 0.5(3h+\sqrt{5}h+2x_0)$. Como $\ell(x)$ se anula en x_0 y x_3, entonces el máximo absoluto de $|\ell(x)|$ es $\text{máx}\{|\ell(r_1)|,|\ell(r_2)|,|\ell(r_3)|\} = \left\{\frac{9h^4}{16}, h^4\right\} = h^4$. Finalmente,

$\therefore$ El error general al interpolar con cuatro puntos igualmente espaciados es $|f(x)-P_3(x)| \leq \frac{M_4h^4}{24}$, $x \in [x_0,x_3]$.

$\therefore$ Si solo interpolamos valores $x \in [x_1,x_2]$, el máximo absoluto de $|\ell(x)|$ en este intervalo se alcanza en el punto medio $x = (x_1+x_3)/2 = 0.5(3h+2x_0)$ y es $\frac{9h^4}{16}$. En este caso la estimación del error general es

$$|f(x)-P_3(x)| \leq \frac{3M_4h^4}{128},\ x \in [x_1,x_2].$$

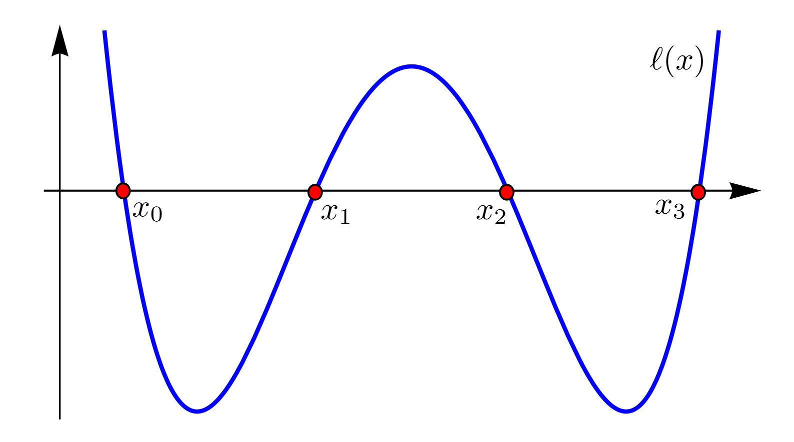


Figura 2.5 $\ell(x) = (x-x_0)(x-x_1)(x-x_2)(x-x_3)$

Ejemplo 2.17

Si tabulamos la función $\operatorname{sen} x$ para $x_0 = 0$, $x_1 = 0.05$, $x_2 = 0.10, x_3 = 0.15, \ldots$ entonces el error al interpolar con P_3 entre x_1 y x_2 es

$$|\operatorname{sen} x - P_3(x)| \leq 1 \cdot \frac{3}{128}(0.05)^4 \approx 1.46 \times 10^{-7},$$

pues $|\operatorname{sen} x| \leq 1\ \forall\, x$ (Aquí suponemos que el polinomio se evalúa de manera exacta).

2.14 Error con interpolación con polinomios de grado n.

Si interpolamos sobre puntos igualmente espaciados $x_i = x_0 + i \cdot h,\ i = 0,1,...,n$; y si h es pequeño entonces $f^{(n+1)}(\xi(x))$ en general no se espera que varíe gran cosa. El comportamiento del error es entonces principalmente determinado por $\ell(x) = (x-x_0)(x-x_1)\cdots(x-x_n)$.

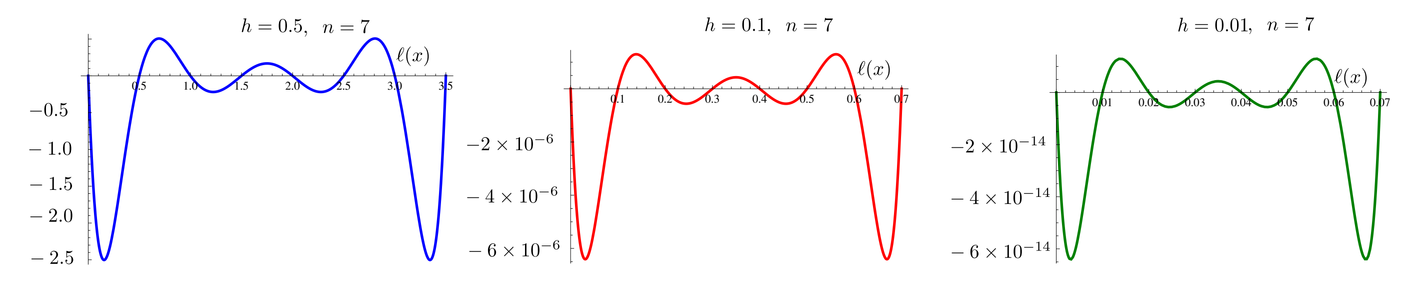


Figura 2.6 $\ell(x) = (x-x_0)(x-x_1)\cdots(x-x_n)$ con $n = 7$.

Pero las oscilaciones de $\ell(x)$ se hacen más violentas si n crece,

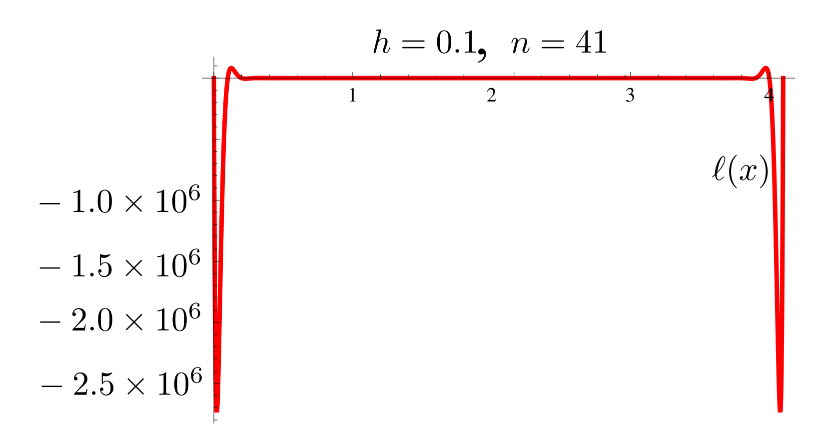


Figura 2.7 $\ell(x) = (x-x_0)(x-x_1)\cdots(x-x_n)$ con $n = 41$.

Sin embargo, la sucesión de polinomios interpolantes $\{P_n(x)\}$ podría converger a f (sin importar si los nodos son o no igualmente espaciados); esto depende del comportamiento de la derivada $k-$ésima de f: La sucesión $\{P_n(x)\}$ converge a f uniformemente en $[a,b]$ (que contiene a los nodos) si

$$\lim_{k\to\infty} \frac{(b-a)^k}{k!} M_k = 0$$

y esto sucede si f es *analítica* en una región suficientemente grande, en el plano complejo, que contenga a $[a,b]$ ([1, pág 84]).

2.15 Otros casos.

Si la función f y sus derivadas son conocidas, se puede hacer una estimación del error con el máximo absoluto.

Ejemplo 2.18

Sea $f(x) = \dfrac{1}{2}e^{(x-1)/2}$. Usando la fórmula de error, estime el error que se cometería al interpolar $f(1)$ con el polinomio interpolante obtenido de la tabla

x	$f(x)$
0.7	0.43
0.8	0.45
1.1	0.53
1.2	0.55

Solución: Son cuatro datos (no igualmente espaciados), $n+1=4$. Luego, la fórmula para estimar el error es

$$|f(1)-P_3(1)| = \left|\frac{f^{(4)}(\xi)}{4!}(1-0.7)(1-0.8)(1-1.1)(1-1.2)\right| \leq \left|\frac{M}{4!}(1-0.7)(1-0.8)(1-1.1)(1-1.2)\right|$$

donde M es el máximo absoluto de $|f^{(4)}(x)| = |\frac{1}{32}e^{\frac{x-1}{2}}|$, en $[0.7, 1.2]$

Cálculo de M

Puntos críticos: La ecuación $f^{(5)}(x) = \frac{1}{64}e^{\frac{x-1}{2}} = 0$ no tiene solución, así que no hay puntos críticos.

Comparación: $M = \text{máx}\{|f^{(4)}(0.7)|, |f^{(4)}(1.2)|\} = 0.0345366...$

Finalmente, la estimación del error es $|f(1)-P_3(1) \leq \left|\dfrac{M}{4!}(1-0.7)(1-0.8)(1-1.1)(1-1.2)\right| = 1.72683\times 10^{-6}$

2.16 Interpolación Iterada de Neville

Si no tenemos información acerca de las derivadas de una función no podemos usar la fórmula para el cálculo del error. Entonces, ¿cuál es el grado del polinomio de interpolación más adecuado para interpolar un valor?. Para responder esta pregunta podemos usar el *algoritmo de Neville,* este método interpola un valor particular con polinomios de grado cada vez más alto (iniciando en grado cero) hasta que los valores sucesivos están suficientemente cercanos. Luego por inspección podemos decidirnos por un valor en particular.

Usemos la siguiente notación: $P_{0,1}$ es el polinomio interpolante que pasa por $(x_0,y_0),(x_1,y_1)$; $P_{0,1,2}$ es el polinomio interpolante que pasa por $(x_0,y_0),(x_1,y_1),(x_2,y_2)$; $P_{1,2,3,4}$ es el polinomio interpolante que pasa por $(x_1,y_1),(x_2,y_2),(x_3,y_3),(x_4,y_4)$; etc. Como no tenemos información acerca de las derivadas de f, el criterio para estimar el error es empírica e implícita: Nos quedamos con la estimación que presente 'menos variación'.

Consideremos la siguiente tabla de datos,

x	1.2	1.3	1.4	1.5	1.6
$f(x)$	0.7651977	0.6200860	0.4554022	0.2818186	0.1103623

Para interpolar en $x = 1.35$ tenemos varias opciones y combinaciones, con tres nodos, con cuatro nodos, etc. Usando nuestra notación, algunos resultados son $P_{0,1,2}(1.35) = 0.5401905$; $P_{123}(1.35) = 0.5388565$; $P_{0123}(1.35) = 0.5395235$; $P_{1234}(1.35) = 0.5395457$; $P_{01234}(1.35) = 0.5395318$. La menor variación la encontramos con $P_{0123}(1.35) = 0.5395235$; $P_{1234}(1.35) = 0.5395457$ y $P_{01234}(1.35) = 0.5395318$ y de estos tres, los más cercanos son $P_{0123}(1.35) = 0.5395235$ y $P_{01234}(1.35) = 0.5395318$. En este caso parece lo mejor quedarnos con la aproximación $P_{01234}(1.35) = 0.5395318$ ya que toma en cuenta toda la tabla.

El problema en el anális anterior es la gran cantidad de polinomios que se deben evaluar, el algoritmo de Neville precisamente automatiza esta tarea usando cálculos anteriores para obtener el nuevo cálculo. El algoritmo de Neville no calcula $P(x)$ sino que *evalúa* varios polinomios interpolantes de Lagrange en un valor dado.

Sea $Q_{i,j}$ el polinomio interpolante que pasa por $(x_{i-j}, y_{i-j}) \ldots (x_i, y_i)$, es decir, $Q_{i,j} = P_{i-j,i-j+1,i-j+2,\ldots,i-1,i}$ es el polinomio interpolante (en la forma de Lagrange) que pasa por los nodos $(x_{i-j}, y_{i-j}), (x_{i-j+1}, y_{i-j+1}), \ldots, (x_i, y_i)$, $0 \le j \le i$. Por ejemplo,

$Q_{0,0} = P_0$ pasa por (x_0, y_0), es decir, $P_0(x_0) = y_0$.

$Q_{4,0} = P_4$ pasa por (x_4, y_4), es decir, $P_4(x_4) = y_4$.

$Q_{5,2} = P_{3,2,1}$ pasa por $(x_3, y_3), (x_4, y_4), (x_5, y_5)$

$Q_{4,4} = P_{0,1,2,3,4}$ pasa por $(x_0, y_0), (x_1, y_1), \ldots, (x_4, y_4)$

Con esta definición de $Q_{i,j}$ se tiene la siguiente relación recursiva,

$$Q_{i,j}(x) = \frac{(x - x_{i-j})Q_{i,j-1}(x) - (x - x_i)Q_{i-1,j-1}(x)}{x_i - x_{j-1}} \tag{2.7}$$

Aplicando esta relación para $i = 1, 2, .., n$; $j = 1, 2, \ldots, i$ se logra calcular varios polinomios interpolantes de Lagrange en un valor x, como se muestra en la siguiente tabla (para el caso de 5 nodos)

x_0	$Q_{0,0} = y_0$				
x_1	$Q_{1,0} = y_1$	$Q_{1,1} = P_{0,1}$			
x_2	$Q_{2,0} = y_2$	$Q_{2,1} = P_{1,2}$	$Q_{2,2} = P_{0,1,2}$		
x_3	$Q_{3,0} = y_3$	$Q_{3,1} = P_{2,3}$	$Q_{3,2} = P_{1,2,3}$	$Q_{3,3} = P_{0,1,2,3}$	
x_4	$Q_{4,0} = y_4$	$Q_{4,1} = P_{3,4}$	$Q_{4,2} = P_{2,3,4}$	$Q_{4,3} = P_{1,2,3,4}$	$Q_{4,4} = P_{0,1,2,3,4}$



http://www.tec-digital.itcr.ac.cr/revistamatematica/cursos-linea/NumericoApplets/Neville.htm

Ejemplo 2.19

La distribución gamma se define como

$$F(x;\beta,\alpha) = \int_0^{x/\beta} \frac{u^{\alpha-1}e^{-u}}{\Gamma(\alpha)}\text{du}$$

Supogamos que tenemos la siguiente tabla de datos, obtenida con $\beta = 1$ y $\alpha = 2$.

x	$F(x;1,2)$
$x_0 = 0$	0.0
$x_1 = 0.1$	0.00467884
$x_2 = 0.2$	0.01752309
$x_3 = 0.3$	0.03693631
$x_4 = 0.4$	0.06155193

Ejemplo 2.19 (continuación).

Si queremos estimar F en 0.25 debemos usar polinomios que al menos pasen por x_2 y x_3. Por ejemplo $P_{0,1,2,3}$, $P_{1,2,3,4}$, etc.

Aplicando el algoritmo de Neville en $x = 0.25$, obtenemos la tabla (redondenado a 7 cifras decimales),

x_0	P_0				
0	0.0				
x_1	P_1	$P_{0,1}$			
0.1	0.0046679	0.0116697			
x_2	P_2	$P_{1,2}$	$P_{0,1,2}$		
0.2	0.0175231	0.0239507	0.0270209		
x_3	P_3	$P_{2,3}$	$P_{1,2,3}$	$P_{0,1,2,3}$	
0.3	0.0369363	0.0272297	0.0264099	0.0265118	
x_4	P_4	$P_{3,4}$	$P_{2,3,4}$	$P_{1,2,3,4}$	$P_{0,1,2,3,4}$
0.4	0.0615519	0.0246285	0.0265794	0.0264947	0.0265011

La menor variación la tenemos entre $P_{1,2,3,4}$ y $P_{0,1,2,3,4}(0.25)$. Como $F(0.25;1,2) = 0.026499021...$, la mejor aproximación en realidad es $P_{1,2,3,4}$, pero en la práctica, por suspuesto, tomamos decisiones sin esta información.

2.16.1 Algoritmo

El algoritmo es muy parecido al método de diferencias divididas de Newton, escribimos la primera columna de la matriz Q (las y_i's) y luego completamos la matriz con la relación recursiva (2.7).

Algoritmo 2.1: Algoritmo de Neville

Datos: Valor a interpolar x y los nodos $(x_0, y_0), (x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$

Salida: Matriz Q

for $i = 0, ..., n$ **do**

  $Q_{i,0} = y_i$

for $i = 1, ..., n$ **do**

  for $j = 1, ..., i$ **do**

$$Q_{i,j}(x) = \frac{(x - x_{i-j})Q_{i,j-1}(x) - (x - x_i)Q_{i-1,j-1}(x)}{x_i - x_{j-1}}$$

return Matriz Q

EJERCICIOS

2.24 Sea $f(x) = x^2 \ln x - x^2$. Supongamos que $P(x)$ es el polinomio interpolante de f obtenido con los datos $(1, -1), (2, -1.2), (3, 0.88)$. Estime el error cometido al aproximar $f(2.71)$ con $P(2.71)$.

2.25 Sea $f(x) = \ln(4x^2 + 2)$ Usando la fórmula de error, estime el error que se cometería al interpolar $f(1.22)$ con el polinomio interpolante obtenido de la tabla

x	$f(x)$
0.5	1.09861
1.1	1.92279
1.2	2.04898
1.3	2.1702

2.26 Considere la tabla de datos

x	e^x
0	1
0.5	$e^{0.5}$
1	e

Estime el error cometido al aproximar $e^{0.6}$ con el polinomio de interpolación correspondiente, en el intervalo $[0, 1]$.

2.27 Sea $f(x) = \cos(3x + 1)$. Supongamos que $P(x)$ es el polinomio interpolante de f obtenido con los datos $(0., 0.54)$, $(0.5, -0.80)$, $(1., -0.65)$. Estime el error cometido al estimar $f(0.71)$ con $P(0.71)$.

2.28 Considere la tabla de datos

x_i	$\cos(1 + 3x_i)$
0	0.540302
1/6	0.070737
1/3	−0.416147

Estime el error cometido al interpolar $\cos(1.75)$ con el polinomio de interpolación obtenido con la tabla anterior, en el intervalo $[0, 1/3]$. **Ayuda:** Observe que en este caso, $x \neq 1.75$!

2.29 Sea $f(x) = \dfrac{1}{2}(\cos x + \operatorname{sen} x)$. Considere el conjunto de puntos $\{(x_i, f(x_i))\}_{i=0,1,2,3}$ con $x_i = i \cdot \pi/2$. Estime el error *general* cometido al aproximar $f(3\pi/4)$ con $P_3(3\pi/4)$.

2.30 Sea $f(x) = \dfrac{x^6}{84} - \dfrac{3\cos(2x)}{8}$. Considere el conjunto de puntos $\{(x_i, f(x_i))\}_{i=0,1,2,3}$ con $x_i = i \cdot 0.2$. Estime el error *general* cometido al aproximar $f(0.65)$ con $P_3(0.65)$.

2.31 Complete la fila 6 en la tabla

x_0	$Q_{0,0} = P_0$				
x_1	$Q_{1,0} = P_1$	$Q_{1,1} = P_{0,1}$			
x_2	$Q_{2,0} = P_2$	$Q_{2,1} = P_{1,2}$	$Q_{2,2} = P_{0,1,2}$		
x_3	$Q_{3,0} = P_3$	$Q_{3,1} = P_{2,3}$	$Q_{3,2} = P_{1,2,3}$	$Q_{3,3} = P_{0,1,2,3}$	
x_4	$Q_{4,0} = P_4$	$Q_{4,1} = P_{3,4}$	$Q_{4,2} = P_{2,3,4}$	$Q_{4,3} = P_{1,2,3,4}$	$Q_{4,4} = P_{0,1,2,3,4}$
x_5	$Q_{5,0} = P_5$	...	...	...	

2.32 Aproximar $F(0.25;1,2)$ (función gamma, ver ejemplo 2.19) usando interpolación lineal, cuadrática y cúbica.

2.33 Use el algoritmo de Neville para aproximar $F(0.25;1,2)$ usando nuestro criterio empírico para obtener una "mejor aproximación".

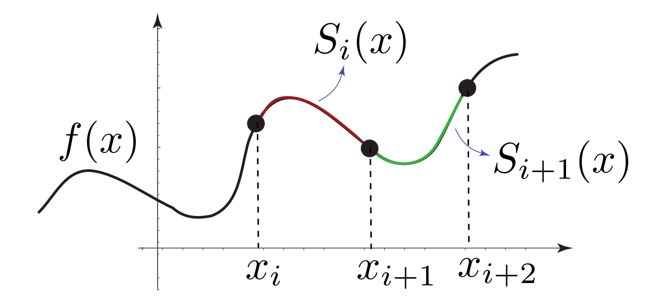
2.34 Supongamos que $x_0, x_1, ..., x_n$ son nodos distintos de un intervalo finito $[a,b]$. Sea $P_n(x)$ el polinomio interpolante obtenido con los datos $\{(x_i, f(x_i)\}_{i=0,1,...,n}$. Si $|f^{(n+1)}(x)| \leq M$ para x en $[a,b]$, muestre que si $x^* \in [a,b]$,

$$|f(x^*) - P_n(x^*)| \leq \frac{M}{(n+1)!}(b-a)^{n+1}$$

2.17 Trazadores Cúbicos (Cubic Splines).

Un trazador (spline) es una banda de hule delgada y flexible que se usa para dibujar curvas suaves a través de un conjunto de puntos. Los trazadores cúbicos (cubic splines) *naturales* se utilizan para crear una función que interpola un conjunto de puntos de datos. Esta función consiste en una unión de polinomios cúbicos, uno para cada intervalo, y está construido para ser una función con derivada primera y segunda continuas. El 'spline' cúbico natural también tiene su segunda derivada igual a cero en la coordenada x del primer punto y el último punto de la tabla de datos.

Supongamos que tenemos $n+1$ puntos $(x_0, y_0), ..., (x_n, y_n)$ con $x_0 < x_1 < ... < x_n$. En vez de interpolar f con un solo polinomio que pase por todos estos puntos, interpolamos la función f en cada subintervalo $[x_k, x_{k+1}]$ con un polinomio cúbico (en realidad de grado ≤ 3) $S_k(x)$ de tal manera que el polinomio cúbico (o trazador cúbico) $S_i(x)$ en $[x_i, x_{i+1}]$ y el trazador cúbico $S_{i+1}(x)$ en $[x_{i+1}, x_{i+2}]$, coincidan en x_{i+1} y que también sus derivadas primera y segunda coincidan en este punto. Cada trazador cúbico coincide con f en los extremos de cada intervalo.



Definición 2.4 (Trazador Cúbico).

Un *trazador cúbico* S es una función a trozos que interpola a f en los $n+1$ puntos (x_0, y_0), (x_1, y_1), (x_2, y_2), ..., (x_n, y_n) (con $a = x_0 < x_1 < ... < x_n = b$). S es definida de la siguiente manera,

$$S(x) = \begin{cases} S_0(x) & \text{si} \quad x \in [x_0, x_1], \\ S_1(x) & \text{si} \quad x \in [x_1, x_2], \\ \vdots & \vdots \\ S_{n-1}(x) & \text{si} \quad x \in [x_{n-1}, x_n], \end{cases}$$

Donde,

(a). $S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$ para $i = 0, 1, \ldots n - 1$

(b). $S(x_i) = y_i$, $i = 0, 1, ..., n$. Para efectos prácticos, $S_j(x_j) = y_j$, $j = 0, 1, ..., n-1$ y $S_{n-1}(x_{n-1}) = y_{n-1}$ y $S_{n-1}(x_n) = y_n$. El siguiente item asegura que $S_j(x_{j+1}) = y_{j+1}$.

(c). $S_i(x_{i+1}) = S_{i+1}(x_{i+1})$ para $i = 0, 1, ..., n-2$

(d). $S_i'(x_{i+1}) = S_{i+1}'(x_{i+1})$ para $i = 0, 1, ..., n-2$

(e). $S_i''(x_{i+1}) = S_{i+1}''(x_{i+1})$ para $i = 0, 1, ..., n-2$

(f). Se satisface una de las dos condiciones que siguen,

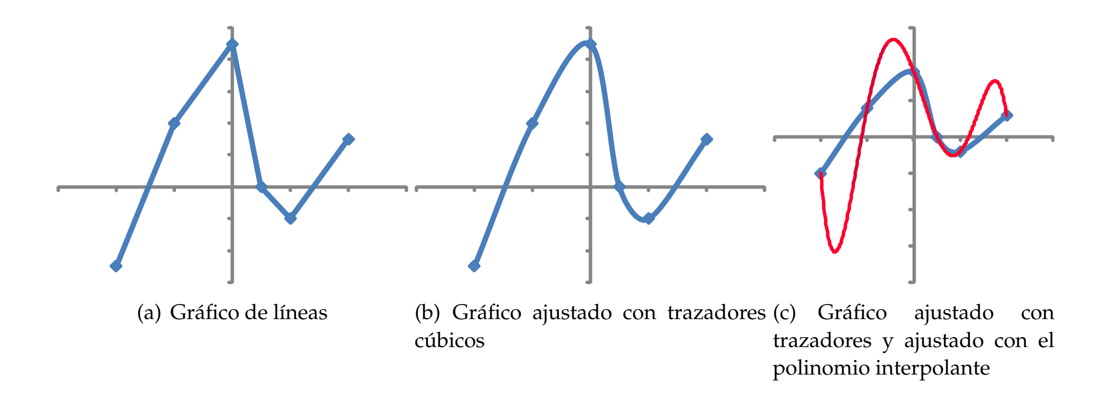
(i). $S''(x_0) = S''(x_n) = 0$ **(frontera libre o natural)**

(ii). $S'(x_0) = f'(x_0)$ y $S'(x_n) = f'(x_n)$ **(frontera sujeta)**

Una aplicación directa de los trazadores cúbicos es la de "suavizar curvas". Tanto en Excel como en *Calc* de OpenOffice o LibreOffice, en las gráficas de dispersión, los pares ordenados (x_i, y_i) se pueden unir con segmentos, con trazadores cúbicos o con el polinomio interpolante (también hay otras opciones, según el modelo o tendencia que se esté aplicando). En la gráfica de la figura (2.17) se muestra un conjunto de datos unidos por segmentos, unidos por trazadores cúbicos y unidos por el polinomio interpolante.

Algunas curvas presentan "picos" así que se construye un trazador para cada curva entre cada dos picos. El tratamiento de picos requiere usualmente un trazador con frontera sujeta.

El proceso de construcción del trazador cúbico consiste en determinar cada polinomio cúbico $S_j(x)$, es decir, buscar sus coeficientes a_i, b_i, c_i y d_i. La definición nos da las condiciones que se deben cumplir. De estas condiciones



(a) Gráfico de líneas

(b) Gráfico ajustado con trazadores cúbicos

(c) Gráfico ajustado con trazadores y ajustado con el polinomio interpolante

podemos obtener un sistema de ecuaciones $4n \times 4n$, donde las incógnitas son todos los coeficientes a_i, b_i, c_i y d_i, $i = 0,1,...,n-1$. Lo que obtenemos es un trazador cúbico único.

Ejemplo 2.20

Determinar el trazador cúbico (frontera libre) para la siguiente tabla,

x_i	$y_i = \cos(3x_i^2)\ln(x_i^3+1)$
$x_0 = 0$	0
$x_1 = 0.75$	-0.0409838
$x_2 = 1.5$	1.31799

Solución: El trazador es, $S(x) = \begin{cases} S_0(x) = a_0 + b_0(x-x_0) + c_0(x-x_0)^2 + d_0(x-x_0)^3 & \text{si} \quad x \in [x_0, x_1], \\ S_1(x) = a_1 + b_1(x-x_1) + c_1(x-x_1)^2 + d_1(x-x_1)^3 & \text{si} \quad x \in [x_1, x_2]. \end{cases}$

Hay que determinar los coeficientes de S_0 y S_1 resolviendo el sistema 8×8,

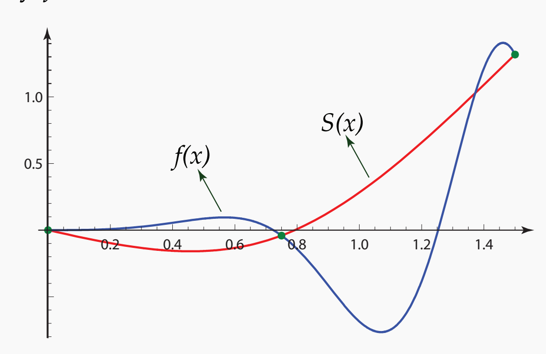
$$\begin{cases} S_0(x_0) &= y_0 \\ S_1(x_1) &= y_1 \\ S_1(x_2) &= y_2 \\ S_0(x_1) &= S_1(x_1) \\ S_0'(x_1) &= S_1'(x_1) \\ S_0''(x_1) &= S_1''(x_1) \\ S_0''(x_0) &= 0 \\ S_1''(x_2) &= 0 \end{cases} \iff \begin{cases} a_0 &= 0 \\ a_1 &= -0.0409838 \\ a_1 + 0.75b_1 + 0.5625c_1 + 0.421875d_1 &= 1.31799 \\ a_0 + 0.75b_0 + 0.5625c_0 + 0.421875d_0 &= a_1 \\ b_0 + 1.5c_0 + 1.6875d_0 &= b_1 \\ 2c_0 + 4.5d_0 &= 2c_1 \\ 2c_0 &= 0 \\ 2c_1 + 4.5d_1 &= 0 \end{cases}$$

La solución de este sistema es $a_0 = 0$, $b_0 = -0.521299$, $c_0 = 0$, $d_0 = 0.829607$, $a_1 = -0.0409838$, $b_1 = 0.878663$, $c_1 = 1.86662$, y $d_1 = -0.829607$. Es decir,

Ejemplo 2.20 (continuación).

$$S(x) = \begin{cases} S_0(x) & = & -0.521299x + 0.829607x^3 \ \text{ si } \ x \in [0,\, 0.75] \\ S_1(x) & = & -0.0409838 + 0.878663(x-0.75) + 1.86662(x-0.75)^2 - 0.829607(x-0.75)^3 \ \text{ si } \ x \in [0.75,\, 1.5]. \end{cases}$$

La representación gráfica para de S y f es



En las figuras (2.8) se muestra el trazador correspondiente a los nodos $x_0 = 0$, $x_1 = 0.5$, $x_2 = 1$, $x_3 = 1.5$ y $x_0 = 0$, $x_1 = 0.375$, $x_2 = 0.75$, $x_3 = 1.125$, $x_4 = 1.5$.

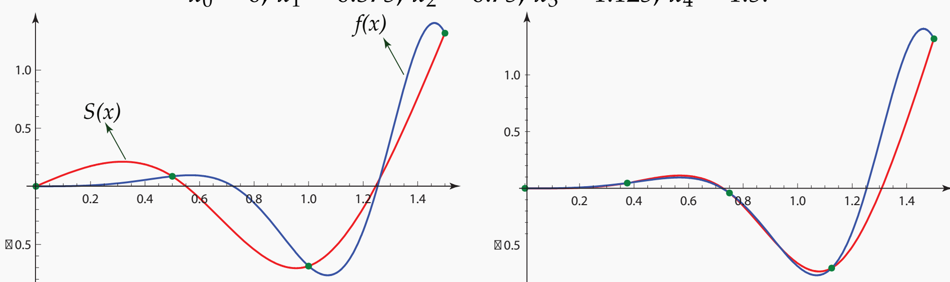
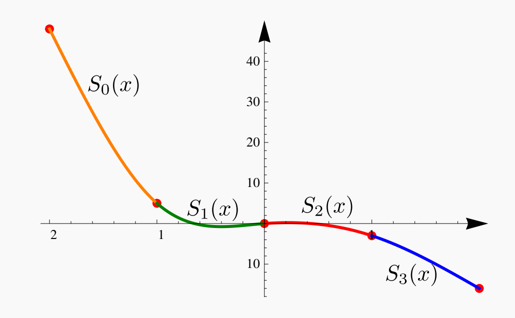


Figura 2.8 Trazador S y f con 3 y 4 puntos

Ejemplo 2.21

Determinar el trazador cúbico (frontera libre) para la siguiente tabla,

x_i	$y_i = x_i^4 - 4x_i^3$
$x_0 = -2$	48
$x_1 = -1$	5
$x_2 = 0$	0
$x_0 = -2$	-3
$x_1 = -1$	-16



Observe que la función $x^4 - 4x^3$ tiene un punto de inflexión en $x = 0$.

Solución: El trazador es,

$$S(x) = \begin{cases} S_0(x) = 9.85714(x+2)^3 - 52.8571(x+2) + 48 & \text{si} \quad x \in [-2,-1], \\ S_1(x) = -11.2857(x+1)^3 + 29.5714(x+1)^2 - 23.2857(x+1) + 5 & \text{si} \quad x \in [-1,0]. \\ S_2(x) = -0.714286x^3 - 4.28571x^2 + 2x & \text{si} \quad x \in [0,1], \\ S_3(x) = 2.14286(x-1)^3 - 6.42857(x-1)^2 - 8.71429(x-1) - 3 & \text{si} \quad x \in [1,22]. \end{cases}$$

Pasos para obtener el trazador cúbico (frontera natural). El proceso general sería como sigue. Sea $h_i = x_{i+1} - x_i$,

- De acuerdo al item **(a)** de la definición (2.4), $S_i(x_i) = y_i \implies a_i = y_i$.

- Haciendo algunas manipulaciones algebraicas en el sistema, se obtiene

$$d_i = \frac{c_{i+1} - c_i}{3h_i} \quad \wedge \quad b_i = \frac{y_{i+1} - y_i}{h_i} - \frac{h_i}{3}(2c_i + c_{i+1}). \tag{2.8}$$

  La condición de frontera natural hace que $c_0 = c_n = 0$.

- Ahora todo depende del cálculo de los c_i's. Éstos se calculan resolviendo el sistema $(n+1) \times (n+1)$

$$\begin{pmatrix} 1 & 0 & 0 & & \cdots 0 \\ h_0 & 2(h_0+h_1) & h_1 & & \cdots 0 \\ 0 & h_1 & 2(h_1+h_2) & h_2 & \cdots 0 \\ & \ddots & \ddots & \ddots & \\ 0 & \cdots & h_{n-3} & 2(h_{n-3}+h_{n-2}) & h_{n-2} \\ 0 & 0 & & & \cdots 1 \end{pmatrix} \cdot \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \\ c_n \end{pmatrix} = \begin{pmatrix} 0 \\ 3(f[x_2,x_1] - f[x_1,x_0]) \\ 3(f[x_3,x_2] - f[x_2,x_1]) \\ \vdots \\ 3(f[x_n,x_{n-1}] - f[x_{n-1},x_{n-2}]) \\ 0 \end{pmatrix}.$$

  Como antes, $f[x_i,x_j] = (y_i - y_j)/(x_i - x_j)$.


Programa en Internet:
http://www.tec-digital.itcr.ac.cr/revistamatematica/cursos-linea/NUMERICO/SitioInterpolacion/SplinesNatural/SplinesNatural.htm

Ejemplo 2.22

Determinar el trazador cúbico (frontera natural) para la siguiente tabla,

x_i	$y_i = \cos(3x_i^2)\ln(x_i^3+1)$
$x_0 = 0$	0
$x_1 = 0.5$	0.0861805
$x_2 = 1$	−0.686211
$x_3 = 1.5$	1.31799

Solución: El trazador es,

$$S(x) = \begin{cases} S_0(x) = a_0 + b_0(x-x_0) + c_0(x-x_0)^2 + d_0(x-x_0)^3 & \text{si} \quad x \in [x_0,x_1], \\ S_1(x) = a_1 + b_1(x-x_1) + c_1(x-x_1)^2 + d_1(x-x_1)^3 & \text{si} \quad x \in [x_1,x_2]. \\ S_2(x) = a_2 + b_2(x-x_1) + c_2(x-x_1)^2 + d_2(x-x_1)^3 & \text{si} \quad x \in [x_2,x_3]. \end{cases}$$

Ejemplo 2.22 (continuación).

Hay que determinar los coeficientes de S_0, S_1 y S_2. Iniciamos calculando los c_i's. Resolvemos el sistema 4×4. Recordemos que $h_i = x_{i+1} - x_i$,

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ h_0 & 2(h_0+h_1) & h_1 & 0 \\ 0 & h_1 & 2(h_1+h_2) & h_2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 3\left(\frac{y_2-y_1}{x_2-x_1} - \frac{y_1-y_0}{x_1-x_0}\right) \\ 3\left(\frac{y_3-y_2}{x_3-x_2} - \frac{y_2-y_1}{x_2-x_1}\right) \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.5 & 2. & 0.5 & 0 \\ 0 & 0.5 & 2. & 0.5 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 0 \\ -5.15143 \\ 16.6596 \\ 0 \end{pmatrix}$$

Obtenemos $c_0 = 0$, $c_1 = -4.96871$, $c_2 = 9.57196$ y, por convenio, el comodín $c_3 = 0$.

Ahora podemos obtener el resto de coeficientes: $a_i = y_i$, los b_i's y los d_i's usando la ecuación (2.8).

$b_0 = 1.00048$, $b_1 = -1.48387$, $b_2 = 0.8177508$
$d_0 = -3.31247$, $d_1 = 9.69378$, $d_2 = -6.38131$.

Finalmente, el trazador cúbico es

$$\begin{cases} S_0(x) = -3.31247x^3 + 0.x^2 + 1.00048x & \text{si} \quad x \in [0, 0.5], \\ S_1(x) = 9.69378(x-0.5)^3 - 4.96871(x-0.5)^2 - 1.48387(x-0.5) + 0.0861805 & \text{si} \quad x \in [0.5, 1], \\ S_2(x) = -6.38131(x-1)^3 + 9.57196(x-1)^2 + 0.8177508(x-1.) - 0.686211 & \text{si} \quad x \in [1, 1.5]. \end{cases}$$

EJERCICIOS

2.35 Calcule el trazador cúbico (natural) para el conjunto de datos $(0,0), (1,1), (2,8)$.

2.36 Considere la tabla de datos,

$T(K)$	100	200	300	400	500	600
$B(cm^3/mol)$	-160	-35	-4.2	9.0	16.9	21.3

Tabla 2.4 Segundos Coeficientes viriales $B(cm^3/mol)$ para el nitrógeno

donde T es la temperatura y B es el segundo coeficiente virial.

a) Calcule el trazador cúbico (natural) para el conjunto de datos de la tabla.

b) ¿Cuál es el segundo coeficiente virial (interpolado) a $450K$?

c) Hacer la representación gráfica del trazador cúbico y del polinomio interpolante $P_5(x)$.

2.37 Considere la siguiente tabla de datos para el agua, donde T es temperatura y ρ es la densidad. Hacer la

$T(C)$	50	60	65	75	80
$\rho(kg/m^3)$	988	985.7	980.5	974.8	971.6

Tabla 2.5

representación gráfica del trazador cúbico y del polinomio interpolante $P_4(x)$.

2.18 Algoritmos e implementación con Basic de OpenOffice o de LibreOffice, y Calc.

2.18.1 Forma de Lagrange del polinomio interpolante

En esta primera implementación calculamos $P_n(x^*)$, es decir, no calculamos el polinomio interpolante; más bien calculamos este polinomio evaluado en un número x^*. Obtener el polinomio es sencillo. Al final de esta subsección se indica cómo hacerlo.

Recordemos que la forma de Lagrange del polinomio interpolante es

$$P_n(x) = y_0 L_{n,0}(x) + y_1 L_{n,1}(x) + ... + y_n L_{n,n}(x)$$

donde

$$L_{n,k}(x^*) = \frac{(x^* - x_0)}{(x_k - x_0)} \cdot \frac{(x^* - x_1)}{(x_k - x_1)} \cdots \frac{(x^* - x_{k-1})}{(x_k - x_{k-1})} \cdot \frac{(x^* - x_{k+1})}{(x_k - x_{k+1})} \cdots \frac{(x^* - x_n)}{(x_k - x_n)}$$

La implementación directa sería acumular el producto 'en rojo', brincar el factor $(x^* - x_k)$ y acumular el producto 'en azul'. El algoritmo sería,

Algoritmo 2.2: Forma de Lagrange del polinomio interpolante - Versión 1.

Datos: $n+1$ datos $\{(x_i, y_i)\}_{i=0,1,...,n}$ con los x_i's distintos; y x^*

Salida: Polinomio interpolante evaluado en x^*, i.e. $P_n(x^*)$

suma$= 0$;
for $k = 0$ **to** n **do**
  $pr = 1$;
  for $i = 0$ **to** n **do**
    if $i \neq k$ **then**
      $pr = \frac{(x^* - x_i)}{(x_k - x_i)} \cdot pr$
  suma= suma + $y_k * pr$
return suma

Una implementación en Basic (OpenOffice y LibreOffice) sería

Código VBA 2.1: Interpolación con forma de Lagrange del polinomio interpolante. Versión 1.

```
Function Lagrange1(X(), y(), xx)
dim k,suma,pr,n,i
```

```
suma = 0
n = UBound(X) rem UBound elije el mas grande subíndice del arreglo
for k=0 to n
     rem cálculo de cada Ln,k(xx)
    pr = 1
    for i = 0 to n
  if i <> k then
       pr = pr*(xx-X(i))/(X(k)-X(i))
  end if
    next i  rem yk*Ln,k(xx)
    suma = suma + Y(k)*pr
next k
Lagrange1 = suma
end function
```

Una implementación eficiente. La implementación que sigue, calcula lo mismo pero con la mitad de operaciones. Consiste en calcular cada $L_{n,k}(x)$ de manera escalonada, construyendo varios polinomios al mismo tiempo con los índices disponibles.

$j=1$	`Ln(0) =` `Ln(1) =`	$(x^*-x_1)/((x_0-x_1))$ $(x^*-x_0)/((x_1-x_0))$

$j=2$	`Ln(0) =`
	`Ln(1) =`
	`Ln(2) =`
		$(x^*-x_1)(x^*-x_2)/((x_0-x_1)(x_0-x_2))$
		$(x^*-x_0)(x-x_2)/((x_1-x_0)(x_1-x_2))$
		$(x^*-x_0)(x-x_1)/((x_2-x_0)(x_2-x_1))$

$j=3$	`Ln(0) =`
	`Ln(1) =`
	`Ln(2) =`
	`Ln(3) =`
		$(x^*-x_1)(x^*-x_2)(x^*-x_3)/((x_0-x_1)(x_0-x_2)(x_0-x_3))$
		$(x^*-x_0)(x^*-x_2)(x^*-x_3)/((x_1-x_0)(x_1-x_2)(x_1-x_3))$
		$(x^*-x_0)(x^*-x_1)(x^*-x_3)/((x_2-x_0)(x_2-x_1)(x_2-x_3))$
		$(x^*-x_0)(x^*-x_1)(x^*-x_2)/((x_3-x_0)(x_3-x_1)(x_3-x_2))$

El algoritmo es,

Algoritmo 2.3: Forma de Lagrange del polinomio interpolante

Datos: $n+1$ datos $\{(x_i,y_i)\}_{i=0,1,\ldots,n}$ con los x_i's distintos; y x^*
Salida: Polinomio interpolante evaluado en x^*, i.e. $P_n(x^*)$

suma= 0;
for $k=0$ **to** n **do**
    $Ln(k)=1$;
for $j=1$ **to** n **do**
    for $k=0$ **to** $j-1$ **do**
        $Ln(k)=(x^*-x_j)/(x_k-x_j)\cdot Ln(k)$
    for $k=0$ **to** $j-1$ **do**
        $Ln(j)=(x^*-x_k)/(x_j-x_k)\cdot Ln(j)$
for $k=0$ **to** n **do**
    suma= suma + $y_k\cdot Ln(k)$
return suma

Como se ve, en el primer `For` anidado k va de 0 a $j-1$ luego j se incrementa (sería el índice $'j=k'$ que debemos brincar en el numerador) y j pasa al denominador, luego se vuelve a incrementar y pasa al numerador, etc.

Una implementación en Basic (OpenOffice y LibreOffice) sería

Código VBA 2.2: Interpolación con forma de Lagrange del polinomio interpolante. Versión 2.

```
Function Lagrange2(X(), Y(), xx)
 Dim suma,j, k, n
 Dim Ln()
   n = UBound(X)
   ReDim Ln(0 to n)

   For k = 0  To n
       Ln(k) = 1
   Next k
   For j = 1  To n
     For k = 0  To j-1
       Ln(k)=(xx-X(j))/(X(k)-X(j))*Ln(k)
     Next k
     For k = 0  To j-1
         Ln(j)=(xx-X(k))/(X(j)-X(k))*Ln(j)
     Next k
  Next j
  For k = 0  To n
       suma= suma+Y(k)*Ln(k)
  Next k
Lagrange2= suma
End Function
```

Cuaderno completo OpenOffice o LibreOffice Las funciones `Lagrange1(X(),Y(),xx)` y `Lagrange2(X(), Y(), xx)`, reciben los vectores $X=(x_0,x_1,...,x_n)$, $Y=(y_0,y_1,...,y_n)$, y el valor a interpolar $x^*=$`xx`. Ambas funciones devuelven $P_n(x^*)$.

La función `Lagrange` la llamamos desde la subrutina `Main`. En esta subrutina usamos una variable `rango` para la selección de datos que hace el usuario. En Basic los rangos inician en 0, por lo que si seleccionamos $n+1$ datos, `n = rango.Rows.getCount()-1`. El vector `X` y el vector `Y` los inicializamos con

```
For i=0 To n
    X(i)=rango.getCellByPosition(0, i).Value
    Y(i)=rango.getCellByPosition(1, i).Value
Next i
```

Par usar esta función, vamos a usar el cuaderno de la figura (2.9). El usuario debe seleccionar una subtabla y hacer clic en el botón. Como ya indicamos, en Basic de OpenOffice o de LibreOffice, este evento lo manejamos así: La selección se recibe en una variable `range`. Luego pasamos la información de este rango a los vectores `X()` e `Y()`. Luego llamamos a la función `Lagrange2(X,Y,xx)`. Observe que no es necesario pasar los valores del rango a los vectores X e Y, solo lo hacemos porque de esta manera la implementación va a la par de la teoría.

La lectura del rango y la lectura del valor x^* la hacemos desde la subrutina `Main`. Una vez leídos estos datos, se llama a la la función `Lagrange2`. El botón tendrá asociada la subrutina `Main`.

	xi	yi	Valor a Interpolar x*	Forma de Lagrange P(x*)	Interpolar_Lagrange
9	0,0000	-0,41615	0,5	0,06250	
10	0,500	0,54030			
11	1,000	1,00000			
12	0,710	0,83646			
13	0,0000	-0,41615			

Figura 2.9 Cuaderno para la implementación de la forma de Lagrange del polinomio interpolante.



Software: Cuadernos LibreOffice y Excel

Código VBA 2.3: Subrutina `Main` del cuaderno `Lagrange`

```
Option Explicit
Sub Main    'subrutina principal
        'Cargar la biblioteca BblMatematica
        BasicLibraries.loadLibrary("BblMatematica" )
        dim i          'contador
        dim rango      'tabla de datos
        dim xx         'valor a interpolar
        dim n          'filas seleccionadas desde 0 hasta n
        dim X(), y()   'xi's y yi's
            'El usuario hace una selección con el ratón
        rango=ThisComponent.getCurrentSelection()
                'número de filas seleccionadas
         n=rango.Rows.getCount()-1        'n+1=número de datos
        if n<1 then
                msgbox "Por favor, seleccione los datos"
                Exit Sub
        end if
        Redim X(0 to n) 'pasamos los datos a un vector por comodidad
        Redim Y(0 to n)
        For i=0 to n
                X(i)=rango.getcellbyposition(0,i).Value
                Y(i)=rango.getcellbyposition(1,i).Value
        next i
        'Interpola (la function 'cells' está en BblMatematica)
        xx = Cells("C9").value
        Cells("D9").Value=Lagrange2(X(),Y(), xx)
End Sub
```

¿Cómo imprimir el polinomio?. Para imprimir el polinomio solo habría que hacer una pequeña modificación. La función `polyLagrange(X,Y)` devuelve un cadena de texto ('String'). Usamos la función `Str()` para convertir los números a cadena de texto y la concatenación del texto se hace con el operador '+'.

Código VBA 2.4: Imprimir el polinomio interpolante en la forma de Lagrange

```
Function polyLagrange(X(), Y())
 Dim suma,j, k, n
 Dim Dn, Nn, Pn 'Variables para el texto
 Dim Ln()
 n = UBound(X)
 ReDim Ln(0 to n,1 to 2)
 For k = 0 To n
     Ln(k,1) =""
     Ln(k,2) =""
 Next k

 For j = 1 To n
    For k = 0 To j-1
       Ln(k,1)=Ln(k,1)+"(x-"+Str(X(j))+")"
       Ln(k,2)=Ln(k,2)+"("+Str(X(k))+"-"+Str(X(j))+")"
    Next k
    For k = 0 To j-1
       Ln(j,1)="(x-"+Str(X(k))+")"+Ln(j,1)
       Ln(j,2)="("+Str(X(j))+"-"+Str(X(k))+")"+Ln(j,2)
    Next k
Next j

For k = 0 To n
     Pn=Pn+"+"+Str(Y(k))+"*"+Ln(k,1)+"/("+Ln(k,2)+")"
Next k
      'Depuración -- = +,+- = - ('ReplaceString' está en BblMatematica)
 Pn=ReplaceString(Pn," + ","- -",)
 Pn=ReplaceString(Pn," - ","+ -")
 PolyLagrange = "Pn(x)= " + Pn
End Function
```

Una corrida con la selección `{(1,-4), (2,5), (3,6)}` devuelve el polinomio

```
P_n(x)=  -4*(x- 2)(x- 3)/(( 1- 2)( 1- 3))
         +5*(x- 1)(x- 3)/(( 2- 1)( 2- 3))
         +6*(x- 2)(x- 1)/(( 3- 2)( 3- 1))
```

Implementación en *MATHEMATICA*. En *MATHEMATICA* podemos escribir el código de manera más directa. El código se puede escribir usando la paleta `"Basic Input"`. El módulo `Lagrange` calcula el polinomio interpolante,

Para hacer una corrida ejecutamos el código

```
Lagrange[XY_] :=
  Module[{j, k, n, X, Y},
   X_k_ := Transpose[XY][[1,k+1]];
   Y_k_ := Transpose[XY][[2,k+1]];
   n = Length[XY] - 1;
   Return[ Sum_{k=0}^{n} Y_k (Prod_{j=0}^{k-1} (x - X_j)/(X_k - X_j)) (Prod_{j=k+1}^{n} (x - X_j)/(X_k - X_j))]];

XY = {-0.1, -0.19}, {0, 0.29}, {0.1, -0.38}};
{P[x] = Lagrange[XY], P[0.35]}
```

La salida es `{-9.5 (-0.1 + x) x - 29. (-0.1 + x) (0.1+ x) -19. x (0.1+ x), -7.08625}.`

EJERCICIOS

2.38 Vamos a usar la tabla del ejemplo (2.2) para interpolar $f(0.25)$ y comparar con el valor correcto $f(0.25) = 0.001200416039457...$

a) Interpolar $f(0.25)$ con tres datos
b) Interpolar $f(0.25)$ con cuatro datos
c) Interpolar $f(0.25)$ con toda la tabla

2.39 En este ejercicio vamos a hacer la representación gráfica de una función f conocida y su polinomio interpolante $P_3(x)$.

a) Calcule la forma de Lagrange del polinomio interpolante $P_3(x)$, usando la tabla de datos que está a la derecha. Hacer la representación gráfica de ambas funciones en $[0,2]$

b) Repita el ejercicio anterior ampliando la tabla a 15 datos con el mismo paso $h = 0.5$.

x	$y = \cos(3x^2)\ln(x^3+1)$
0	0
0.5	0.0861805
1	-0.686211
1.5	1.31799

2.40 Considere la función de Runge, $f(x) = \dfrac{1}{1+25x^2}$.

a) Considere el conjunto de datos

$$x_i = -1 + i \cdot \frac{2}{n}, \; y_i = f(x_i), \; i = 0,1,...,n \subseteq [-1,1].$$

Hacer la representación gráfica de los polinomios interpolantes para $n = 5,10,20$. Represente estos polinomios conjuntamente con f.

2.41 **(Nodos de TChebyshev).** Usando la función de Runge y los datos $\{(x_i, f(x_i))\}_{i=0,1,...,n} \subseteq [-1,1]$ con $x_i = \cos\left(\dfrac{2i+1}{2n+2}\pi\right)$, calcule los polinomios interpolantes para $n = 3,8,20$ y represente estos polinomios conjuntamente con f.

2.19 Forma modificada y forma baricéntrica de Lagrange.

La implementación se centra en el cálculo de los ω_k. Una vez calculados estos números, armar cada polinomio interpolante es algo directo. Recordemos que

$$P_n(x) = \ell(x) \sum_{k=0}^{n} \frac{\omega_k}{x - x_k} y_k$$

donde $\omega_k = \dfrac{1}{(x_k - x_1)(x_k - x_2)...(x_k - x_{k-1})(x_k - x_{k+1})...(x_k - x_n)}.$

- En el algoritmo, para calcular cada ω_k, separamos el denominador en dos factores

$$(x_k - x_1)(x_k - x_2)...(x_k - x_{k-1})(x_k - x_{k+1})...(x_k - x_n).$$

  El primer factor y el segundo factor en el denominador de cada uno de los ω_k's se calcula con el ciclo

```
w_k = 1,   para cada   k = 0,...,n
For j = 1 To n
    For k = 0 To j-1
        w_k  =  (x_k - x_j) w_k
    For k = 0 To j-1
        w_j  =  (x_j - x_k) w_j
```

  En este ciclo, el primer `For` anidado produce el segundo factor en el denominador de cada uno de los ω_k's,

  $w_0 = (x_0 - x_1)(x_0 - x_2)\cdots(x_0 - x_n)$,
  $w_1 = (x_1 - x_2)(x_1 - x_3)\cdots(x_1 - x_n)$,
  $w_2 = (x_2 - x_3)(x_2 - x_4)\cdots(x_2 - x_n)$,

  y el segundo `For` anidado completa el producto,

  $w_1 = (x_1 - x_0)(x_1 - x_2)(x_1 - x_3)\cdots(x_1 - x_n)$,
  $w_2 = (x_2 - x_0)(x_2 - x_1)(x_2 - x_3)(x_2 - x_4)\cdots(x_2 - x_n)$, etc.

  Finalmente, $\omega_k = 1/w_k$.

- Si usamos nodos de TChebyshev, el cálculo es directo: $\omega_k = (-1)^k \operatorname{sen} \dfrac{(2k+1)\pi}{2n+2}$. Este último resultado se obtiene del siguiente cálculo:

$$\begin{aligned}
\omega_k^{-1} &= \lim_{x \to x_k} \frac{\ell(x)}{(x - x_k)} \\
&= \lim_{x \to x_k} \frac{\ell(x) - \ell(k)}{(x - x_k)}, \quad \text{pues} \quad \ell(x_k) = 0; \\
&= \ell'(x_k).
\end{aligned}$$

Algoritmo 2.4: Cálculo de los Pesos Baricéntricos

Datos: $n+1$ nodos distintos $\{x_i\}_{i=0,1,...,n}$.
Salida: Pesos baricéntricos ω_k, $k=0,1,...,n$

if $\{x_i\}_{i=0,...,n}$ *son nodos de TChebyshev* **then**
  $w_k = (-1)^k \operatorname{sen} \dfrac{(2k+1)\pi}{2n+2}$, $k=0,...,n$
else
  for $k=0$ **to** n **do**
    $w_k = 1$
  for $j=1$ **to** n **do**
    for $k=0$ **to** $j-1$ **do**
      $w_k = (x_k - x_j)w_k$
    for $k=0$ **to** $j-1$ **do**
      $w_j = (x_j - x_k)w_j$
  for $k=0$ **to** n **do**
    $w_k = 1/w_k$
return $w_0, w_1, ..., w_n$

Implementación en Basic OpenOffice o LibreOffice. Aquí solo implementamos la función `ModificadaLagrange(X,Y,x*)`. Esta subrutina recibe los vectores $X=(x_0,x_1,...,x_n)$, $Y=(y_0,y_1,...,y_n)$, y el valor a interpolar `x*`. Esta función devuelve $P_n(x^*)$. El código de esta función sería,

Código VBA 2.5: Polinomio Interpolante. Forma Modificada de Lagrange

```
Function  ModificadaLagrange(X(),Y(), xval)
Dim suma,j, k, n,L
Dim W()
 n = UBound(X)
 ReDim W(0 to n)
 For k = 0 To n
    W(k) = 1
 Next k
 For j = 1 To n
     For k = 0 To j-1
        W(k)=(X(k)-X(j))*W(k)
     Next k
     For k = 0 To j-1
        W(j)=(X(j)-X(k))*W(j)
     Next k
 Next j

 For k = 0 To n
    W(k)=1/W(k)
 Next k
```

```
 L=1
 For k = 0 To n
    L=(xval-X(k))*L
 Next k


 For k = 0 To n
    suma = suma+Y(k)*W(k)/(xval-X(k))
 Next k
ModificadaLagrange = L*suma
End Function
```

EJERCICIOS

2.42 Agregar la forma modificada de Lagrange al cuaderno que contiene la función `Lagrange`.

2.43 Implementar la forma baricéntrica de Lagrange en el caso general y el caso de nodos igualmente espaciados.

2.20 Forma de Newton del polinomio interpolante.

La forma de Newton del polinomio interpolante es

$$\begin{aligned} P(x) &= f[x_0] + f[x_0,x_1](x-x_0) + f[x_0,x_1,x_2](x-x_0)(x-x_1) \\ &\quad + \cdots + f[x_0,...,x_n](x-x_0)\cdots(x-x_{n-1}), \end{aligned}$$

donde los coeficientes están en la diagonal de la matriz de diferencias divididas,

$$\begin{array}{lllll} y_0 = f[x_0] & & & & \\ y_1 & f[x_0,x_1] & & & \\ y_2 & f[x_1,x_2] & f[x_0,x_1,x_2] & & \\ y_3 & f[x_2,x_3] & f[x_1,x_2,x_3] & & \\ \vdots & \vdots & \vdots & \ddots & \\ y_n & f[x_{n-1},x_n] & f[x_{n-2},x_{n-1},x_n] & \cdots & f[x_0,x_1,...,x_n] \end{array}$$

Para la implementación de la fórmula de diferencias divididas de Newton es conveniente reescribir el polinomio como

$$\begin{aligned} P(x) &= C_{0,0} + C_{1,1}(x-x_0) + C_{2,2}(x-x_0)(x-x_1) \\ &\quad + \cdots + C_{n,n}(x-x_0)\cdots(x-x_{n-1}), \end{aligned}$$

y la matriz de diferencias divididas como

$$\begin{array}{lllll}
y_0 = C_{0,0} & & & & \\
y_1 & C_{1,1} & & & \\
y_2 & C_{2,1} & C_{2,2} & & \\
y_3 & C_{3,1} & C_{3,2} & & \\
\vdots & \vdots & \vdots & \ddots & \\
y_n & C_{n,1} & C_{n,2} & \cdots & C_{n,n}
\end{array}$$

Para el cálculo de los $C_{i,i}$'s usamos la fórmula recursiva:

$$\begin{aligned}
C_{i,0} &= y_i, \;\; i = 0,2,...,n \\
C_{i,j} &= \frac{C_{i,j-1} - C_{i-1,j-1}}{x_i - x_{i-j}}, \quad j = 1,2,...n, \;\; i = j,2,...,n
\end{aligned} \tag{2.9}$$

Algoritmo 2.5: Diferencias Divididas de Newton

Datos: $\{(x_i, y_i)\}_{i=0,1,...,n}$ con los x_i's distintos.
Salida: Coeficientes del polinomio interpolante: $C_{0,0}, C_{1,1}, ..., C_{n,n}$

```
for i = 1 to n do
    C_{i,0} = y_i
;
for j = 1 to n do
    for i = j to n do
        C_{i,j} = (C_{i,j-1} - C_{i-1,j-1}) / (x_i - x_{i-j})
return C_{0,0}, C_{1,1}, ..., C_{n,n}
```

Para evaluar el polinomio interpolante en x^* se requiere los coeficientes $C_{j,j}$ y los nodos $x_0, ...x_n$.

Algoritmo 2.6: Evaluar la forma de Newton del polinomio interpolante

Datos: Nodos $x_0, x_1, ..., x)n$, coeficientes $C_0, C_1, ..., C_n$ y x^*
Salida: $P_n(x^*)$

```
suma = C_0;
factor = 1;
for j = 1 to n do
    factor=factor·(x* − x_{j−1});
    suma=suma+C_j·factor;
return suma
```

Implementación en Basic de OpenOffice o LibreOffice. Las funciones `DDNewton` usa dos vectores `C1` y `C2`. En `C1` se almacena la columna actual y en `C2` la nueva columna.Cada elemento de la diagonal `C2(j)` se almacena en el vector `Cij`. Para recalcular `C2` se actualiza `C1`. En vez de poner `C1=C2` se usa un `For` para hacer la copia componente a componente (en las nuevas versiones de OpenOffice y LibreOffice esto puede haber cambiado y se podría poner directamente `C1=C2` de manera segura).

Código VBA 2.6: Matriz de diferencias divididas de Newton e Interpolación

```
Function  DDNewton(X(), Y()) As Variant
  Dim  i, j, n,k
  Dim C1() As double, C2() As double, Cij() as double

  n = UBound(X)
  ReDim C1(0 to n)
  ReDim C2(0 to n)
  ReDim Cij(0 to n)
  'continua en la pág siguiente

  For i = 0 To n
    C1(i) = Y(i)
  Next i
  Cij(0)=Y(0)
  For j = 1 To n
     For i = j To n
        'Calcula la columna j
         C2(i) = (C1(i) - C1(i - 1)) / (X(i) - X(i - j))
     Next i
     Cij(j)=C2(j)
        'Actualiza C1
     For k = 0 To n
         C1(k) = C2(k)
     Next k
  Next j
  DDNewton = Cij()
 End Function
'------------------------------------------------------------------
Function  EvalDDNewton(Cij(),X(), xx) As Double
 Dim j, n, suma, factor
 suma=Cij(0)
 factor=1
 n = UBound(Cij)
  For j = 1 To n
     factor=factor*(xx-X(j-1))
     suma = suma+Cij(j)*factor
  Next j
  EvalDDNewton=suma
End Function
```

Cuaderno DDNewton Como en la implementación de la forma de Lagrange, aquí también tenemos los datos en dos vectores `X` e `Y`. La función `DDNewton(X,Y)` devuelve un vector con la diagonal de la matriz de diferencias divididas: $C_0, C_1, ..., C_n$. Para evaluar la forma de Newton implementamos la función `EvalDDNewton(C(),X(),x*)` usando el algoritmo (2.6). También se incluye la función `PolyDDNewton(C(),X())` que devuelve el polinomio $P_n(x)$ y la subrutina `MatrizDDNewton(X(), Y(),columna, fila)` para imprimir la matriz de diferencias divididas desde la celda `(columna,fila)`.

Para ejecutar estas macros, usamos como referencia el cuaderno de la figura (2.10).

			Valor a Interpolar	Forma de Newton	Interpolar_DDNewton		
8	xi	yi	x*	P(x*)			
9	1,0000	1,00000	0,5	0,25000			
10	0,000	0,00000					
11	-1,000	1,00000			Pn(x) = 1+ 1*(x - 1)+ 1*(x - 1)(x - 0)		
12	2,000	16,00000					
13	3,0000	81,00000			1,00000		
14					0,00000	1,00000	
15					1,00000	-1,00000	1,00000

Figura 2.10 Cuaderno para la implementación de la forma de Newton del polinomio interpolante.

El botón ejecuta la subrutina `Main`.



Software: Cuadernos LibreOffice y Excel

Código VBA 2.7: Subrutina `Main` del cuaderno DDNewton

```
Sub Main  REM subrutina principal
        'Cargar la biblioteca BblMatematica
        BasicLibraries.loadLibrary("BblMatematica" )

    rem function cells está en BblMatematica
    dim i          'contador
        dim rango     'tabla de datos
        dim xx        'valor a interpolar
        dim n         'filas seleccionadas desde n=0 hasta n
        dim X(), y(), C()  'xi's y yi's, C recibe la diagonal de matriz de diferencias divididas

            'El usuario hace una selección con el ratón
        rango=ThisComponent.getCurrentSelection()
                   'número de filas seleccionadas
         n=rango.Rows.getCount()-1        'n+1=numero de datos

        if n<1 then
                msgbox "Por favor, seleccione los datos"
                Exit Sub
        end if

        Redim X(0 to n) 'pasamos los datos a un vector por comodidad
    Redim Y(0 to n)
    ReDim C(0 To n)
        For i=0 to n
                X(i)=rango.getcellbyposition(0,i).Value
                Y(i)=rango.getcellbyposition(1,i).Value
        next i
        'Leer xx
         xx=Cells("C9").Value
        'C recibe la diagonal de la matriz de diferencias divididas
    C= DDNewton(X, Y)
         'Interpolar f(xx). la función 'Cells' está en BblMatematica
    Cells("D9").Value = EvalDDNewton(C,X, xx)
```

```
        'imprimir Pn(x)
    Cells("E11").String = PolyDDNewton(C,X)
        'imprimir la matriz
    Call MatrizDDNewton(X, Y,4,12) 'Iniciar en "E13"

End Sub
```

Para imprimir la matriz de diferencias divididas se modifica la función `DDNewton`.

Código VBA 2.8: Imprimir matriz de diferencias divididas y el polinomio en la forma de Newton

```
Sub MatrizDDNewton(X(), Y(),columna, fila)
 Dim  i, j,n, k, Hoja
 Dim C1(), C2(), Cij()
 n = UBound(X)

 ReDim C1(0 to n)
 ReDim C2(0 to n)
 ReDim Cij(0 to n)

 For i = 0 To n
  C1(i) = Y(i)   'Imprime primera columna. 'Cells' está en Bblmatematica
  Cells(columna, fila+i).Value=C1(i)
 Next i
 Cij(0)=Y(0)
 For j = 1 To n
  For i = j To n 'Imprime la columna j
   C2(i) = (C1(i) - C1(i - 1)) / (X(i) - X(i - j))
   Cells(columna+j, fila+i).Value=C2(i)
  Next i
   Cij(j)=C2(j)
   For k = 0 To n
    C1(k) = C2(k)
   Next k
 Next j
 End Sub

'Imprimir polinomio en la forma de Newton  ------------------------------
 Function PolyDDNewton(Cij(),X()) As String
 Dim j,n
 Dim Pn, factor
 Pn= Str(Cij(0))
 factor=""
 n = UBound(Cij)
 For j = 1 To n
     factor=factor+"(x -"+Str(X(j-1))+")"
     Pn = Pn+"+"+Str(Cij(j))+"*"+factor
  Next j
          'Reemplazar -- = +,+- = -. 'ReplaceString' está en Bblmatematica
```

```
  Pn=ReplaceString(Pn," + "," - -",)
  Pn=ReplaceString(Pn," - ","+ -")
  PolyDDNewton="Pn(x) = " + Pn
End Function
```

EJERCICIOS

2.44 Vamos a usar la tabla (I.1, pág 3) para interpolar $f(0.25)$ y comparar con el valor correcto $f(0.25) = 0.001200416039457...$

a) Interpolar $f(0.25)$ con tres datos
b) Interpolar $f(0.25)$ con cuatro datos
c) Interpolar $f(0.25)$ con toda la tabla

2.45 En este ejercicio vamos a hacer la representación gráfica de una función f conocida y su polinomio interpolante $P_3(x)$.

a) Calcule la forma de Newton del polinomio interpolante $P_3(x)$, usa-ndo la tabla de datos que está a la derecha. Hacer la representación gráfica de ambas funciones en $[0,2]$
b) Repita el ejercicio anterior ampliando la tabla a 15 datos.

x	$y = \cos(3x^2)\ln(x^3+1)$
0	0
0.5	0.0861805
1	-0.686211
1.5	1.31799

2.46 Considere la función de Runge, $f(x) = \dfrac{1}{1+25x^2}$.

a) Considere el conjunto de datos

$$x_i = -1 \ + \ i\cdot\frac{2}{n},\ y_i = f(x_i),\ i = 0,1,...,n \ \subseteq \ [-1,1].$$

Hacer la representación gráfica de los polinomios interpolantes para $n = 5,10,20$. Represente estos polinomios conjuntamente con f.

b) **(Nodos de Chebyshev).** Usando los datos $\{(x_i, f(x_i))\}_{i=0,1,...,n} \subseteq [-1,1]$ con $x_i = \cos\left(\dfrac{2i+1}{2n+2}\pi\right)$, calcule los polinomios interpolantes para $n = 3,8,20$ y represente estos polinomios conjuntamente con f.

2.21 Trazadores cúbicos

Recordemos que un *trazador cúbico* S es una función a trozos que interpola a f en los $n+1$ puntos (x_0,y_0), (x_1,y_1), (x_2,y_2), ..., (x_n,y_n) (con $a = x_0 < x_1 < ... < x_n = b$). S es definida de la siguiente manera,

$$S(x) = \begin{cases} S_0(x) & \text{si} \quad x \in [x_0,x_1], \\ S_1(x) & \text{si} \quad x \in [x_1,x_2], \\ \quad\vdots & \quad\vdots \\ S_{n-1}(x) & \text{si} \quad x \in [x_{n-1},x_n], \end{cases}$$

Donde, $S_i(x) = a_i + b_i(x-x_i) + c_i(x-x_i)^2 + d_i(x-x_i)^3$ para $i = 0,1,\ldots n-1$.

Para el cálculo de los coeficientes a_i, b_i, c_i y d_i usamos el algoritmo

Algoritmo 2.7: Trazador cúbico con frontera natural

Datos: $(x_0, y_0), (x_1, y_1), \ldots (x_n, y_n)$ con $x_0 < x_1 < \cdots < x_n$.
Salida: Coeficientes a_j, b_j, c_j, d_j, para cada $j = 0, 1, \ldots n-1$.

for $j = 0, 1, \ldots n-1$ **do**
  $h_j = x_{j+1} - x_j$;
  $a_j = y_j$;
$a_n = y_n$;
for $j = 1, \ldots n-1$ **do**
  $\alpha_j = \dfrac{3}{h_j} \cdot (a_{j+1} - a_j) - \dfrac{3}{h_{j-1}} \cdot (a_j - a_{j-1})$.
$l_0 = 1$, $\mu_0 = 0$, $z_0 = 0$;
$l_n = 1$, $z_n = 0$, $c_n = 0$;
for $i = 1, 2, \ldots, n-1$ **do**
  $l_i = 2(x_{i+1} - x_{i-1}) - h_{i-1}\mu_{i-1}$;
  $\mu_i = h_i / l_i$;
  $z_i = (\alpha_i - h_{i-1} z_{i-1}) / l_i$;
for $j = n-1, n-2, \ldots, 0$ **do**
  $c_j = z_j - \mu_j c_{j+1}$;
  $b_j = (a_{j+1} - a_j)/h_j - h_j(c_{j+1} + 2c_j)/3$;
  $d_j = (c_{j+1} - c_j)/(3h_j)$;
return a_j, b_j, c_j, d_j, $j = 0, 1, \ldots n-1$.

Implementación en Basic con OpenOffice o LibreOffice. La función `TrazadorCubico(X(), Y())` recibe los datos en dos vectores `X`, `Y` y devuelve una matriz con los coeficientes de las polinomios $S_i(x)$.

Código VBA 2.9: Trazadores Cúbicos (Cubic Splines)

```
Function TrazadorCubico(X(), Y(), xx,co,fi)
       Dim i, j, n, suma
       Dim a(), b(), c(), d(), h(), l(), mu(), z(), alfa()
       Dim S()

       n = UBound(X)

       ReDim a(0 to n)
       ReDim b(0 to n)
       ReDim c(0 to n)
       ReDim d(0 to n)
       ReDim h(0 to n)
       ReDim l(0 to n)
       ReDim mu(0 to n)
       ReDim z(0 to n)
       ReDim alfa(0 to n)
       Redim S(0 to n-1, 0 to 3)
```

```
For j = 0 To n - 1
        h(j) = X(j + 1) - X(j)
        a(j) = Y(j)
Next j

a(n) = Y(n)

For j = 1 To n - 1
        alfa(j) = 3 / h(j) * (a(j + 1) - a(j)) - 3 / h(j - 1) * (a(j) - a(j - 1))
Next j

l(0) = 1
mu(0) = 0
z(0) = 0
l(n) = 1
z(n) = 0
c(n) = 0

For i = 1 To n - 1

        l(i) = 2 * (X(i + 1) - X(i - 1)) - h(i - 1) * mu(i - 1)
        mu(i) = h(i) / l(i)
        z(i) = (alfa(i) - h(i - 1) * z(i - 1)) / l(i)

Next i

For j = n - 1 To 0 Step -1

        c(j) = z(j) - mu(j) * c(j + 1)
        b(j) = (a(j + 1) - a(j)) / h(j) - h(j) * (c(j + 1) + 2 * c(j)) / 3
        d(j) = (c(j + 1) - c(j)) / (3 * h(j))

Next j

'Coeficientes
For j = 0 To n - 1
        S(j,0)=a(j)
        S(j,1)=b(j)
        S(j,2)=c(j)
        S(j,3)=d(j)
Next j
               'Calcula valor interpolado con el Sj(x) adecuado.
For j = 0 To n - 1
        If X(j)<=xx AND xx<=X(j+1) Then
                suma = S(j,0)+S(j,1)*(xx-X(j))+S(j,2)*((xx-X(j))^2)+S(j,3)*((xx-X(j))^3)
        End If
Next j
```

```
  TrazadorCubico=suma
End Function
```

Para ejecutar estas macros, usamos como referencia el cuaderno de la figura (2.11).

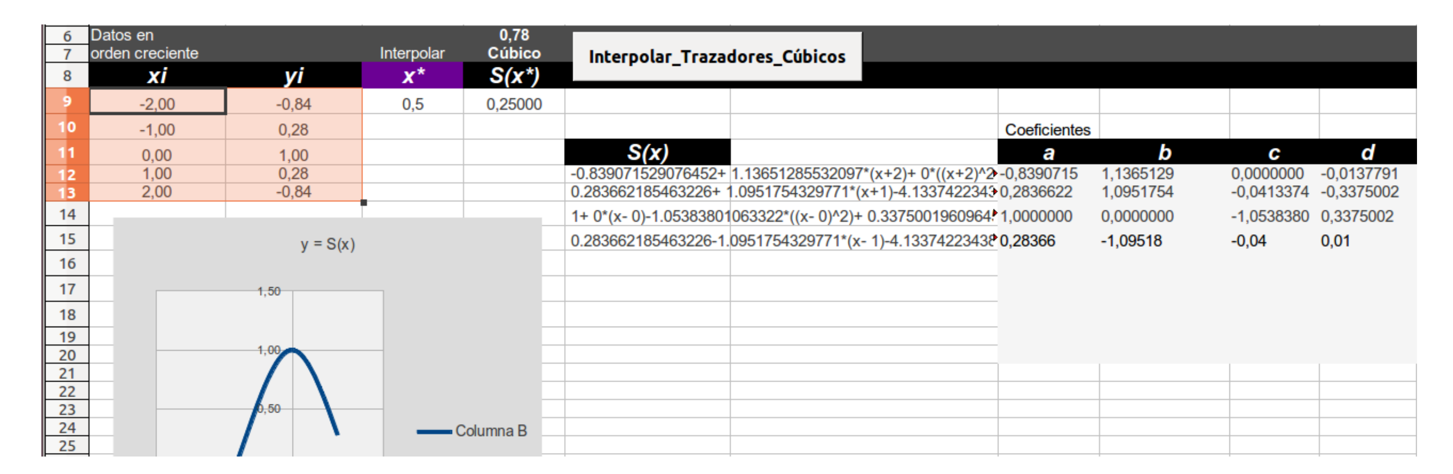


Figura 2.11 Cuaderno para la implementación de el trazador cúbico.

El botón ejecuta la subrutina `Main`.

 **Software:** Cuadernos LibreOffice y Excel

Código VBA 2.10: Subrutina `'Main'` del cuaderno `TrazadorCúbico`

```
Option Explicit
Sub Main
    'Cargar biblioteca de funciones especiales: Cells, ReplaceString, CleanRange, etc.
    BasicLibraries.loadLibrary("BblMatematica" )

        Dim columna, fila
        Dim rango
        Dim X(), Y(),S()
        Dim n, i, j, xx

        'Rango selecionado por el usuario
        rango = ThisComponent.getCurrentSelection()
        n = rango.Rows.getCount()-1 'n+1=número de datos

        'Limpiar celdas de cálculo anterior
        CleanRange (4, 11, 10)

        'CleanRange (20, 20 , 40)
        If n<2 Then
                MsgBox "Por favor, seleccione los datos."
                Exit Sub
        End If
```

```
        ReDim X(0 To n)
        ReDim Y(0 To n)
        Redim S(0 to n-1, 0 to 3) 'Coeficientes del trazador cúbico

        For i=0 To n
                X(i)=rango.getCellByPosition(0, i).Value
                Y(i)=rango.getCellByPosition(1, i).Value
        Next i

        xx = Cells("C9").Value

                        'Interpolar con Sj(xx). Imprime coeficinets en columna, fila
        columna = 6 : fila=11 'desde G12
        Cells("D6").Value = TrazadorCubico(X, Y, xx, columna, fila)

           'Imprimir S(x) desde columna = 4 : fila=11 'desde E12
        columna = 4 : fila=11 'desde E12
        ImprimirPolinomioTrazadorCubico(X(), Y(), columna, fila)
          'Graficar S(x). Los datos están anclados en  "A9"
          'Grafica los datos con la opción CubicSplines
        GraficarTrazadorCubico(0,8, n)
End Sub
```

EJERCICIOS

2.47 Implementar una función para imprimir cada uno de los polinomios $S_i(x)$.

2.48 Implementar una función para hacer la representación gráfica de $S(x)$ (Ver apéndice).

2.49 Verifique su implementación con el ejemplo (2.22).

2.50 Implementar una función para evaluar $S(x^*)$.

2.51 Usar la implementación para interpolar $f(0.27)$ y $f(0.554)$; usando la tabla

x_i	y_i
$x_0 = 0$	0.656
$x_1 = 0.2$	-0.086
$x_2 = 0.4$	0.68
$x_3 = 0.6$	-1.799



Versión más reciente (y actualizaciones) de este libro:
http://www.tec-digital.itcr.ac.cr/revistamatematica/Libros/
http://dl.dropbox.com/u/57684129/revistamatematica/Libros/index.html

3 INTERPOLACIÓN. ASPECTOS TEÓRICOS.

Un problema de interpolación polinomial se especifica como sigue: dados $n+1$ puntos $(x_0,y_0),(x_1,y_1),...,(x_n,y_n)$, siendo todos los x_i's distintos, encontrar un polinomio $P_n(x)$ de grado $\leq n$ tal que

$$P_n(x_i) = y_i,\ \ i = 0,1,2,...,n \tag{3.1}$$

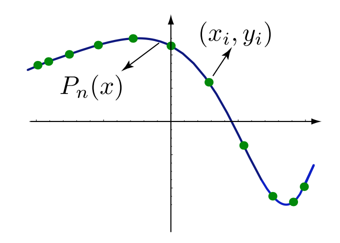


Figura 3.1 $P_n(x)$

Teorema 3.1 (Existencia y unicidad).

Sean $x_0,x_1,...,x_n$ números reales distintos y $n \geq 0$. Si $y_0,y_1,...,y_n$ son números reales arbitrarios, existe un único polinomio P_n, de grado menor o igual a n tal que $P_n(x_i) = y_i$, $i = 0,1,...,n$.

Prueba. Unicidad: Sean P_n y Q_n son dos polinomios interpolantes de grado a lo sumo n tales que $P_n(x_i) = Q_n(x_i) = y_i$, $i = 0,1,2,...,n$. Entonces el polinomio $R(x) = P_n(x) - Q_n(x)$ es de grado a lo sumo n y tiene a lo sumo n ceros o es el polinomio nulo (teorema fundamental del algebra). Como $R(x)$ tiene $n+1$ ceros, debe ser el polinomio nulo, es decir, $P_n(x) \equiv Q_n(x)$.

Existencia: La prueba es por inducción.

Para $n = 0$ existe un polinomio P_0 de grado a lo sumo 0 tal que $P_0(x_0) = y_0$. Por supuesto, se trata del polinomio nulo.

Supongamos que el resultado es cierto para $n-1 \geq 0$, es decir, existe P_{n-1} de grado a lo sumo $n-1$ tal que $P_{n-1}(x_i) = y_i$, $i = 0,1,...,n-1$. Consideremos el polinomio

$$P_n(x) = P_{n-1}(x) + a_n\,(x-x_0)(x-x_1)...(x-x_{n-1}),\ \ a_n \text{ una constante.}$$

Como $\prod_{j=0}^{n-1}(x-x_j)$ es de grado n, P_n tiene grado a lo sumo n (algún coeficiente podría ser nulo). Solo falta verificar que se puede elegir la constante a_n tal que $P_n(x_i)=y_i$, $i=0,1,...,n$. En efecto, por hipótesis

$$P_n(x_i)=P_{n-1}(x_i)\ +\ 0\ =\ y_i\ \text{ si }\ i=0,1,...,n-1.$$

Ahora, evaluando P_n en x_n se prueba que existe una constante a_n adecuada para que $P_n(x_n)=y_n$,

$$P_{n-1}(x_n)+a_n\prod_{j=0}^{n-1}(x_n-x_j)\ =\ y_n\ \Longrightarrow\ a_n=\frac{y_n-P_{n-1}(x_n)}{\prod_{j=0}^{n-1}(x_n-x_j).}$$

Observe que $\prod_{j=0}^{n-1}(x_n-x_j)\neq 0.$

Existen varias formas para el polinomio interpolante. $\mathbb{R}_{n+1}[x]$ es el conjunto de polinomios con coeficientes reales, de grado $\leq n$ y tiene dimensión $n+1$. Si $\Omega=\{B_0,B_2,...,B_n\}$ es una base para $\mathbb{R}_{n+1}[x]$, entonces el polinomio interpolante P_n se puede escribir en la base Ω como

$$P_n(x)=a_1B_0(x)+...+a_nB_n(x)$$

Aunque el polinomio interpolante es único, la forma (antes de simplificar) puede cambiar según la base que se tome. Si tomamos la base $\{1,x,...,x^n\}$ de $\mathbb{R}_{n+1}[x]$, entonces

$$P_n(x)=a_0+a_1x+...+a_nx^n$$

Como $P_n(x_i)=y_i$, $i=0,1,...,n$, tenemos el sistema de ecuaciones lineales

$$\begin{pmatrix} 1 & x_0 & x_0^2 & ... & x_0^n \\ 1 & x_1 & x_1^2 & ... & x_1^n \\ \vdots & \vdots & & \cdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix}\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix}=\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix} \tag{3.2}$$

La matriz asociada V_n del sistema 3.2 se llama matriz de Vandermonde. Las columnas de V_n conforman un conjunto de vectores linealmente independiente (ejercicio 3.1). Por lo tanto $\text{Det}(V_n)\neq 0$. Además (ver 3.2),

$$\text{Det}(V_n)=\prod_{0\leq i<j\leq n}(x_i-x_j)$$

Por ejemplo,

$$\text{Det}(V_3)=(x_0-x_1)(x_0-x_2)(x_1-x_2)(x_0-x_3)(x_1-x_3)(x_2-x_3)$$

Los coeficientes $a_0,a_1,...,a_n$ se pueden obtener por regla de Cramer. El cálculo del polinomio interpolante con este método es muy costoso, además de que los coeficientes a_i no siempre se calculan tan exactos como uno quisiera

(resolviendo el sistema).

Ejemplo 3.1

Para los datos $(x_0, y_0), (x_1, y_1)$ la matriz de Vandermonde es $\begin{pmatrix} 1 & x_0 \\ 1 & x_1 \end{pmatrix}$

Usando la regla de Cramer, $P(x) = \dfrac{x_1 y_0 - x_0 y_1}{x_1 - x_0} + \dfrac{y_1 - y_0}{x_1 - x_0} x.$

3.1 Forma de Lagrange para el polinomio interpolante.

La forma de Lagrange del polinomio interpolante se obtiene usando la base $L_{n,0}(x), ..., L_{n,n}(x)$ donde $L_{n,k}(x) = \prod_{\substack{i=0 \\ i \neq k}}^{n} \dfrac{x - x_i}{x_k - x_i}$. En este caso

$$P_n(x) = a_0 L_{n,0}(x) + a_1 L_{n,1}(x) + ... + a_n L_{n,n}(x)$$

Como $P_n(x_i) = y_i$, $i = 0, 1, ..., n$ y como $L_{n,j}(x_i) = \delta_{ij}$ (delta de Kronecker), tenemos el sistema de ecuaciones lineales

$$\begin{pmatrix} 1 & 0 & 0 & ... & 0 \\ 0 & 1 & 0 & ... & 0 \\ \vdots & \vdots & & \cdots & \vdots \\ 0 & 0 & & \cdots & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix} \quad (3.3)$$

de donde $a_i = y_i$, $i = 0, 1, ..., n$.

3.2 Forma de Lagrange modificada y forma baricéntrica de Lagrange.

La forma de Lagrange del polinomio interpolante es atractiva para propósitos teóricos. Sin embargo se puede reescribir en una forma que se vuelva eficiente para el cálculo computacional. Una exposición más extensa se encuentra en ([2])

Forma Modificada.

$$P_n(x) = \ell(x) \sum_{j=0}^{n} \frac{\omega_j^{(\mathrm{n})}}{x - x_j} y_j \quad (3.4)$$

Forma Baricéntrica.

$$P_n(x) \begin{cases} = \ y_i & \text{si} \quad x = x_i, \\ \\ = \dfrac{\displaystyle\sum_{k=0}^{n} \frac{\omega_k^{(n)}}{x - x_k} y_k}{\displaystyle\sum_{k=0}^{n} \frac{\omega_k^{(n)}}{x - x_k}}, & \text{si} \quad x \neq x_i \end{cases}$$

$\ell(x)$ y $\omega_k^{(n)}$ se definen así: Supongamos que tenemos $n+1$ nodos distintos $x_0, x_1, ..., x_n$. $\ell(x) = \prod_{i=0}^{n} (x - x_i)$ y definimos los *pesos baricéntricos* como

$$\omega_0^{(0)} = 1, \ \omega_k^{(n)} = \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{1}{x_i - x_k}.$$

La forma modificada se obtiene notando que $L_{n,k}(x) = \dfrac{\omega_k^{(n)}}{x - x_k} \prod_{j=0}^{n} (x - x_j) = \dfrac{\omega_k^{(n)}}{x - x_k} \ell(x)$.

La forma baricéntrica se obtiene dividiendo el polinomio interpolante por $\sum_{k=0}^{n} L_{n,k}(x) \equiv 1$ (es decir, el polinomio interpolante de $f(x) \equiv 1$ es $P_n(x) \equiv 1$), en efecto

$$\begin{aligned} P_n(x) &= \sum_{k=0}^{n} y_k L_{n,k}(x) = \frac{\displaystyle\sum_{k=0}^{n} y_k L_{n,k}(x)}{\displaystyle\sum_{k=0}^{n} L_{n,k}(x)} \\ &= \frac{\displaystyle\sum_{k=0}^{n} y_k \frac{\omega_k^{(n)}}{x - x_k} \prod_{j=0}^{n} (x - x_j)}{\displaystyle\sum_{k=0}^{n} \frac{\omega_k^{(n)}}{x - x_k} \prod_{j=0}^{n} (x - x_j)} = \frac{\displaystyle\sum_{k=0}^{n} \frac{\omega_k^{(n)}}{x - x_k} y_k}{\displaystyle\sum_{k=0}^{n} \frac{\omega_k^{(n)}}{x - x_k}} \quad \text{si} \quad x \neq x_k. \end{aligned}$$

Esta forma se llama "forma baricéntrica" de Lagrange pues, aunque no todos los "pesos" son necesariamente positivos, expresa este polinomio como un promedio ponderado de los y_k's.

Nodos de TChebyshev.

En el caso de que podamos escoger los nodos, la elección son los nodos de TChebyshev. En este caso el cálculo de los pesos baricéntricos es muy sencillo ([2], p.249),

$$\omega_k^{(n)} = (-1)^k \operatorname{sen} \frac{(2k+1)\pi}{2n+2}$$

3.3 Forma de Newton para el polinomio interpolante.

La forma de Newton del polinomio interpolante se obtiene usando la base $B_0(x),...,B_n(x)$ donde

$$\begin{array}{rcl} B_0(x) & = & 1 \\ B_1(x) & = & (x-x_0) \\ B_2(x) & = & (x-x_0)(x-x_1) \\ \vdots & & \vdots \\ B_n(x) & = & (x-x_0)(x-x_1)\cdots(x-x_{n-1}) \end{array}$$

Tenemos entonces

$$P_n(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + \cdots + a_n(x-x_0)\cdots(x-x_{n-1}),$$

Como $P_n(x_i) = y_i,\ i = 0,1,...,n$ tenemos

$$\begin{array}{rcll} P_n(x_0) & = & a_0 & = y_0 \\ P_n(x_1) & = & a_0 + a_1(x_1-x_0) & = y_1 \\ P_n(x_2) & = & a_0 + a_1(x_2-x_0) + a_2(x_2-x_0)(x_2-x_1) & = y_2 \\ \vdots & & \vdots & \\ P_n(x_n) & = & a_0 + a_1(x_n-x_0) + ... + a_n(x_n-x_0)\cdots(x_n-x_{n-1}) & = y_n \end{array}$$

Como el sistema es triangular, podemos despejar directamente cada a_i. Para obtener una fórmula recursiva un poco más limpia, definimos $Q_0 = a_0$ y

$$Q_j(x) = a_0 + a_1(x-x_0) + ... + a_j(x-x_0)\cdots(x-x_{j-1}),\ \ j = 1,...,n-1$$

De esta manera,

$$\begin{array}{rcl} a_0 & = & y_0 \\ a_1 & = & \dfrac{y_1 - Q_0(x_1)}{x_1 - x_0} \\ a_2 & = & \dfrac{y_2 - Q_1(x_2)}{(x_2-x_0)(x_2-x_1)} \\ \vdots & & \vdots \\ a_n & = & \dfrac{y_n - Q_{n-1}(x_n)}{(x_n-x_0)\cdots(x_n-x_{n-1})} \end{array}$$

Obtenemos una fórmula recursiva: $a_0 = y_0$ y $a_k = \dfrac{y_k - Q_{k-1}(x_k)}{(x_k - x_0)\cdots(x_k - x_{k-1})}$, $k = 1, ..., n$

Diferencias divididas. Si $y_k = f(x_k)$, la fórmula anterior nos muestra que cada a_k depende de $x_0, x_1, ..., x_k$. Desde muchos años atrás se usa la notación $a_n = f[x_0, x_1, ...x_n]$ para significar esta dependencia.

Al símbolo $f[x_0, x_1, ...x_n]$ se le llama *diferencia divida* de f.

Si consideramos $f[x_0, x_1, ...x_n]$ como una función de $n+1$ variables, entonces es una función *simétrica,* es decir, permutar las variables de cualquier manera no afecta el valor de la función. Esto es así porque el polinomio que interpola los puntos $\{(x_i, y_i)\}_{i=0,...,n}$ es único, por lo tanto sin importar el orden en que vengan los puntos, el coeficiente principal siempre es $a_n = f[x_0, x_1, ...x_n]$.

El nombre *diferencia divida* viene de la agradable propiedad

Teorema 3.2

La diferencia dividida $f[x_0, x_1, ...x_n]$ satisface la ecuación

$$f[x_0, x_1, ...x_n] = \frac{f[x_1, x_2, ...x_n] - f[x_0, x_1, ...x_{n-1}]}{x_n - x_0} \tag{3.5}$$

Prueba. Sea $P_k(x)$ el polinomio que interpola f en $x_0, x_1, ..., x_k$. Aquí solo necesitamos $P_n(x)$ y $P_{n-1}(x)$. Sea $R(x)$ el polinomio que interpola f en $x_1, x_2, ..., x_n$. Entonces (ejercicio 2.1)

$$P_n(x) = R(x) + \frac{x - x_n}{x_n - x_0}[R(x) - P_{n-1}(x)] \tag{3.6}$$

Como en la ecuación (3.6) el polinomio de la izquierda y el de la derecha son idénticos, entonces su coeficiente principal debe ser el mismo, es decir,

$$\begin{aligned}
f[x_0, x_1, ...x_n]x^n + ... &= f[x_1, x_2, ...x_n]x^{n-1} + ... + \frac{x\,R(x) - x\,P_{n-1}(x) + ...}{x_n - x_0} \\
&= f[x_1, x_2, ...x_n]x^{n-1} + \\
&\quad ... + \frac{f[x_1, x_2, ...x_n]x^n + ... - f[x_0, x_2, ...x_{n-1}]x^n + ...}{x_n - x_0} \\
&= \frac{(f[x_1, x_2, ...x_n] - f[x_0, x_2, ...x_{n-1}])x^n}{x_n - x_0} + ...
\end{aligned}$$

de donde se obtiene (3.5).

Interpolando sobre $\{(x_i, y_i)\}_{i=k-j,...,k}$ tenemos

$$f[x_{k-j}, x_{k-j+1}, \cdots, x_k] = \frac{f[x_{k-j+1}, ..., x_k] - f[x_{k-j}, x_1, ..., x_{k-1}]}{x_k - x_{k-j}}$$

Resumiendo: En notación de diferencias divididas,

$$\begin{aligned} a_0 &= y_0, \\ a_1 &= f[x_0, x_1], \\ a_2 &= f[x_0, x_1, x_2] \\ \vdots & \quad \vdots \\ a_n &= f[x_0, x_1, ..., x_n]. \end{aligned}$$

donde, $f[x_k] = y_k$ y, en general,

$$f[x_{k-j}, x_{k-j+1}, \cdots, x_k] = \frac{f[x_{k-j+1}, ..., x_k] - f[x_{k-j}, x_1, ..., x_{k-1}]}{x_k - x_{k-j}}$$

En particular,

$$\begin{aligned} f[x_i, x_j] &= \frac{y_i - y_j}{x_i - x_j} \\ f[x_{k-2}, x_{k-1}, x_k] &= \frac{f[x_{k-1}, x_k] - f[x_{k-2}, x_{k-1}]}{x_k - x_{k-2}} \\ f[x_{k-3}, x_{k-2}, x_{k-1}, x_k] &= \frac{f[x_{k-2}, x_{k-1}, x_k] - f[x_{k-3}, x_{k-2}, x_{k-1}]}{x_k - x_{k-3}} \\ &\vdots \\ f[x_0, x_1, ..., x_k] &= \frac{f[x_1, x_2, ..., x_k] - f[x_0, x_1, ..., x_{k-1}]}{x_k - x_0} \end{aligned}$$

Este esquema recursivo se puede arreglar en forma matricial como sigue,

$$\begin{array}{llll} y_0 & = \boldsymbol{a_0} & & \\ y_1 & f[x_0, x_1] = \boldsymbol{a_1} & & \\ y_2 & f[x_1, x_2] & f[x_0, x_1, x_2] = \boldsymbol{a_2} & \\ & f[x_2, x_3] & f[x_1, x_2, x_3] & \\ \vdots & \vdots & \vdots & \ddots \\ y_n & f[x_{n-1}, x_n] & f[x_{n-2}, x_{n-1}, x_n] & f[x_0, x_1, ..., x_n] = \boldsymbol{a_n} \end{array}$$

3.4 Estimación del error.

Sea $P_n(x)$ el polinomio que interpola f en los puntos $x_0, x_1, ..., x_n$. La fórmula de error debe ser exacta para $x = x_i$, $i = 0, 1, ..., n$ y también debe ser exacta en el caso de que f sea un polinomio de grado inferior o igual a n. Esto sugiere que en la fórmula de error deben aparecer los factores $(x - x_0)(x - x_1)\cdots(x - x_n)$ y f^{n+1}. Probando con $f(x) = x^{n+1}$ se observa que debe aparecer el factor $1/(n+1)!$.

Teorema 3.3

Sea $f \in C^{n+1}[a,b]$. Sea $P_n(x)$ el polinomio de grado $\leq n$ que interpola f en los $n+1$ puntos (distintos) $x_0, x_1, ..., x_n$ en el intervalo $[a,b]$. Para cada valor fijo $x \in [a,b]$ existe $\xi(x) \in]a,b[$ tal que

$$f(x) - P_n(x) = \frac{f^{n+1}(\xi(x))}{(n+1)!}(x - x_0)(x - x_1)\cdots(x - x_n)$$

Prueba. Este es un razonamiento muy elegante, debido a Cauchy[6]. Si $x = x_i$, $i = 0, 1, ...n$, la fórmula de error es correcta. Consideremos un valor fijo $x \in [a,b]$, diferente de cada uno de los nodos x_i, $i = 0, 1, ..., n$. Definamos una función g, en la variable t, de la siguiente manera,

$$g(t) = f(t) - P_n(t) - \frac{f(x) - P_n(x)}{\displaystyle\prod_{i=0}^{n}(x - x_i)} \prod_{i=0}^{n}(t - x_i)$$

$g \in C^{n+1}[a,b]$ y $g(x) = 0$ y $g(x_i) = 0$, $i = 0, 1, ...n$.

Por tanto g tiene $n+2$ ceros distintos en $[a,b]$. El teorema de Rolle dice que si h es continua en $[a,b]$ y derivable en $]a,b[$, y si $h(a) = h(b) = 0$, entonces $h(\varepsilon) = 0$ para algún $\varepsilon \in]a,b[$. Aplicando repetidamente el teorema de Rolle a la función g en los intervalos $[x_0, x_1]$, $[x_1, x_2]$, ..., concluimos que g' tiene *al menos* $n+1$ ceros distintos en $]a,b[$. De manera similar concluimos que

g'' tiene *al menos* n ceros distintos en $]a,b[$

g''' tiene *al menos* $n-1$ ceros distintos en $]a,b[$

$\vdots$

g^{n+1} tiene *al menos* 1 cero distinto en $]a,b[$ (g^{n+1} es continua en [a,b]).

Ahora bien, sea $\xi(x)$ un cero de g^{n+1} en $]a,b[$. Como $\left.\dfrac{d^{n+1}g}{dt^{n+1}}\right|_{t=\xi(x)} = 0$, tenemos (ejercicio 3.4)

6



Agustín Louis Cauchy (1789-1857), padre del análisis moderno. Estableció las bases del análisis matemático basándolo en un concepto riguroso de límite. Fue el creador del análisis complejo. También trabajo en ecuaciones diferenciales, geometría, álgebra, teoría de números, probabilidad y física matemática

$$0 = f^{n+1}(\xi(x)) \; - \; \frac{f(x) \; - \; P_n(x)}{\displaystyle\prod_{i=0}^{n}(x-x_i)}(n+1)!$$

de donde, $f(x) - P_n(x) = \dfrac{f^{n+1}(\xi(x))}{(n+1)!}(x-x_0)(x-x_1)\cdots(x-x_n)$.

3.5 Polinomios de TChebyshev y convergencia.

En la fórmula de error,

$$f(x) - P_n(x) = \frac{f^{n+1}(\xi(x))}{(n+1)!}(x-x_0)(x-x_1)\cdots(x-x_n)$$

el factor $(x-x_0)(x-x_1)\cdots(x-x_n)$ puede ser "optimizado" escogiendo de manera adecuada los nodos $x_0, x_1, ..., x_n$. El proceso de optimización lleva, de manera natural, a un sistema de polinomios llamados polinomios de TChebyshev

El resultado principal es: Para cualquier polinomio mónico $P_n(x)$ de grado n,

$$||P_n(x)||_\infty \geq ||\overline{T}_n(x)||_\infty = \frac{1}{2^{n-1}}, \;\; n \geq 1$$

donde $\overline{T}_n(x)$ es el polinomio mónico de TChebyshev de grado n.

$\overline{T}_{n+1}(x) = (x-x_0)(x-x_1)\cdots(x-x_n)$ con $x_i = \cos\left(\dfrac{2i+1}{2n+2}\,\pi\right), \;\; i = 0,1,....n$

Luego, si P_n interpola f, usando nodos de TChebyshev, tenemos

$$||f(x) - P_n(x)||_\infty \leq \frac{||f^{n+1}||_\infty}{(n+1)!}\,\frac{1}{2^n}$$

Por lo tanto, la convergencia se puede asegurar si la familia de derivadas $f^{(n)}$ es uniformemente acotada ($\exists M$ tal que $\forall\, n \in \mathbb{N}$, $||f^{(n+1)}(x)||_\infty \leq M$). En realidad se necesita menos para asegurar la convergencia!.

Los Polinomios de TChebyshev (de primera especie) se definen, de manera recursiva, de la siguiente manera:

$$\begin{cases} T_0(x) \; = \; 1, \;\; T_1(x) \; = \; x \\ \\ T_{n+1}(x) \; = \; 2xT_n(x) - T_{n-1}(x), \;\; n \geq 1. \end{cases} \tag{3.7}$$

Por ejemplo,

$$\begin{cases} T_2(x) &=& 2x^2-1 \\ T_3(x) &=& 4x^3-3x \\ T_4(x) &=& 8x^4-8x^2+1 \end{cases}$$

Teorema 3.4

Si $x \in [-1,1]$ entonces $T_n(x) = \cos\left(n\cos^{-1}x\right),\ n \geq 0$

Prueba. La idea de la prueba esta basada en el hecho de que $\cos(n\theta)$ es un polinomio en $\cos\theta$, es decir, $\cos n\theta = \sum_{k=0}^{n} a_k \cos^k\theta$. Luego $f_n(\cos\theta) = \cos(n\theta)$ satisface (3.7).

Recordemos que $\cos(a+b) = \cos a\cos b - \operatorname{sen} a \operatorname{sen} b$. Luego,

$$\cos[(n+1)\theta] = \cos\theta\cos(n\theta) - \operatorname{sen}\theta\operatorname{sen}(n\theta).$$
$$\cos(n-1)\theta = \cos\theta\cos(n\theta) + \operatorname{sen}\theta\operatorname{sen}(n\theta).$$

sumando miembro a miembro y reacomodando,

$$\cos[(n+1)\theta] = 2\cos\theta\cos(n\theta) - \cos(n-1)\theta \tag{3.8}$$

Ahora, sea $\theta = \cos^{-1}x$ y $x = \cos\theta$ y $f_n(x) = \cos(n\cos^{-1}x)$. Usando la ecuación (3.8) obtenemos que,

$$\begin{cases} f_0(x) \;=\; 1,\; f_1(x) \;=\; x \\ \\ f_{n+1}(x) \;=\; 2xf_n(x) - f_{n-1}(x),\; n \geq 1. \end{cases}$$

Luego, $f_n = T_n$ para todo n.

- $\overline{T}_n(x) = 2^{n-1}T_n(x)$ **es mónico:** Si $n \geq 1$, de (3.7) se deduce que el coeficiente principal de $T_n(x)$ es 2^{n-1} (ejercicio 3.11). Luego $2^{1-n}T_n(x)$ es mónico.

$$\overline{T}_n(x) = 2^{1-n}T_n(x) \quad \text{y} \quad \overline{T}_0(x) = T_0(x)$$

- **Los ceros de** $\overline{T}_{n+1}(x)$: Si $x \in [-1,1]$ podemos poner $x = \cos\theta$. Luego, $T_{n+1}(\cos\theta) = \cos(n+1)\theta$, así que los ceros $x_k^{(n+1)}$ de T_{n+1} en $[-1,1]$ se pueden obtener resolviendo $\cos[(n+1)\theta] = 0$.

  $\cos[(n+1)\theta] = 0 \implies [(n+1)\theta] = (2k-1)\pi/2,\ \ k \in \mathbb{Z}$. Como $x = \arccos\theta,\ 0 \leq \theta \leq \pi$, entonces

$$x_k^{(n+1)} = \cos\theta_k^{(n+1)},\ \ \theta_k^{(n+1)} = \frac{(2k-1)\pi}{2n+2},\ k = 1,2,\ldots,n+1$$

Por lo tanto, todos los ceros de T_{n+1} son reales, distintos y están contenidos en el intervalo $]-1,1[$.

Ahora podemos escribir

$$\overline{T}_{n+1}(x) = \prod_{k=0}^{n} \left(x - x_k^{(n+1)}\right)$$

- **Extremos:** θ crece de 0 hasta π, por tanto $x = \cos\theta$ decrece de 1 hasta -1. Luego T_n oscila entre 1 y -1. En el ejercicio (3.12) se pide mostrar que

$$T_n \text{ alcanza sus valores extremos en } y_k^{(n)} = \cos\left(\frac{k\pi}{n}\right),\; k = 0,1,...n. \tag{3.9}$$

$$T_n\left(\cos\phi_k^{(n)}\right) = (-1)^k \quad \text{si} \quad \phi_k^{(n)} = \frac{k\pi}{n},\; k = 0,2,...n. \tag{3.10}$$

$$(-1)^k\overline{T}_n(y_k^{(n)}) = \frac{1}{2^{n-1}} \quad \text{y} \quad ||\overline{T}_n(x)||_\infty = \frac{1}{2^{n-1}} \tag{3.11}$$

Los polinomios de TChebyshev son útiles e importantes debido al siguiente teorema

Teorema 3.5

Para cualquier polinomio mónico P_n de grado n

$$||P_n(x)||_\infty \geq ||\overline{T}_n(x)||_\infty = \frac{1}{2^{n-1}}$$

Prueba. La prueba es por contradicción. Supongamos que hay un polinomio P_n mónico de grado n para el que

$$||P_n(x)||_\infty < \frac{1}{2^{n-1}} \tag{3.12}$$

Sea $x_k = \cos(k\pi/n)$. El polinomio $\overline{T}_n$ es mónico de grado n. Entonces de (3.11),

$$(-1)^k P_n(x_k) \leq |P_n(x_k)| < \frac{1}{2^{n-1}} = (-1)^k\overline{T}_n(x_k),\; k = 0,1,...,n.$$

Luego $0 < (-1)^k(\overline{T}_n(x_k) - P_n(x_k)),\; k = 0,1,...,n.$

Esto dice que el polinomio $\overline{T}_n - P_n$ cambia de signo al menos n veces en $[-1,1]$ y hay un cero por cada cambio de signo!

Pero como $\overline{T}_n$ y P_n son mónicos de grado n, $\overline{T}_n - P_n$ tiene grado a lo sumo $n-1$ y entonces no puede tener más de n ceros.

Luego, $\overline{T}_n = P_n$ pero esto contradice (3.12).

El teorema nos dice que si tomamos los $n+1$ ceros de $T_{n+1}(x)$ con los nodos x_i, entonces

$$||f(x) - P_n(x)||_\infty \quad \leq \quad \frac{||f^{n+1}||_\infty}{(n+1)!}||||\overline{T}_{n+1}||_\infty \;=\; \frac{||f^{n+1}||_\infty}{(n+1)!}||\,\frac{1}{2^n}$$

Se puede probar (ver [7]) que si usamos los nodos de TChebyshev, $P_n(x) \longrightarrow f(x)$ si $n \longrightarrow \infty$ uniformemente en $[-1,1]$ con solo que $f \in C^1[a,b]$.

EJERCICIOS

3.1 Use el teorema fundamental del álgebra para probar que las columnas de V_n conforman un conjunto de vectores linealmente independiente.

3.2 Pruebe, usando inducción y operaciones elementales de matrices, que

$$\text{Det}(V_n) = \prod_{0 \leq i < j \leq n} (x_i - x_j)$$

3.3 Probar (3.6). Primero pruebe el caso $i = 1,...,k-1$ y luego los casos $i = 0$ e $i = k$.

3.4 Si $h(t) = \prod_{i=0}^{n}(t - x_i)$, muestre que $h^{(n+1)(t)} = (n+1)!$

3.5 Sea P un polinomio de grado a lo sumo n que interpola una función f en los $n+1$ nodos distintos $x_0, x_1, ...x_n$. Sea t un punto diferente de los nodos anteriores. Sea Q el polinomio de grado a lo sumo $n+1$ que interpola f en $x_0, x_1, ...x_n, t$. Mostrar que

a) $Q(x) = P(x) + f[x_0, x_1, ..., x_n, t]\prod_{j=0}^{n}(x - x_j)$

b) $f(t) - P(t) = f[x_0, x_1, ..., x_n, t]\prod_{j=0}^{n}(t - x_j)$

3.6 Sea $f \in C^n[a,b]$ y sean $x_0, x_1, ...x_n$ puntos distintos en $[a,b]$. Muestre, usando el ejercicio (3.5) y la fórmula de error, que existe un ξ en $]a,b[$ tal que

$$f[x_0, x_1, ..., x_n] = \frac{1}{n!} f^{(n)}(\xi)$$

Ayuda: Compare $f(t)-P(t)$ tal y como aparece en la fórmula del error y como aparece en el ejercicio (3.5).

3.7 Pruebe que si f es un polinomio de grado k, entonces para $n>k$ se tiene

$$f[x_0,x_1,...,x_n]=0$$

3.8 Muestre que si $f(x)=\operatorname{sen}(x)$ es aproximada por un polinomio P de grado nueve que interpola f en diez puntos de $[0,1]$, entonces

$$|\operatorname{sen}(x)-P(x)|\leq \frac{1}{10!},\ x\in[0,1].$$

3.9 Sea $h=x_1-x_0>0$, muestre que si P_1 interpola f en $\{x_0,x_1\}$, entonces

$$||f(x)-P_1(x)||_\infty \leq \frac{||f''||_\infty}{8}\,h^2,\ x\in[x_0,x_1]$$

3.10 Muestre que si P_2 interpola f en $\{x_0,x_0+h,x_0+2h\}, h>0$, entonces

$$||f(x)-P_2(x)||_\infty \leq \frac{||f'''||_\infty}{9\sqrt{3}}\,h^3,\ x\in[x_0,x_0+2h]$$

Ayuda: Observe que $Q(x)=(x-x_0)(x-(x_0+h))(x-(x_0+2h))$ es un polinomio de grado tres. Sus puntos críticos son ceros de la cuadrática Q. Resulta sencillo determinar los extremos absolutos de Q.

3.11 Muestre que el coeficiente principal de $T_n(x)$ es 2^{n-1}.

3.12 Si $x\in[-1,1]$ y $x=\cos\theta,\ T_n(\cos\theta)=\cos(n\theta)$. Use este hecho para verificar que

a) T_n alcanza sus valores extremos en $y_k^{(n)}=\cos\left(\frac{k\pi}{n}\right),\ k=0,1,...n.$

b) $T_n\left(\cos\phi_k^{(n)}\right)=(-1)^k$ si $\phi_k^{(n)}=\frac{k\pi}{n},\ k=0,2,...n.$

c) $(-1)^k\overline{T}_n(y_k^{(n)})=\frac{1}{2^{n-1}}$ y $||\overline{T}_n(x)||_\infty=\frac{1}{2^{n-1}}$

3.13 Considere la función de Runge,

$$f(x)=\frac{1}{1+25x^2}$$

Haga la representación gráfica de $(x-x_0)(x-x_1)...(x-x_n)$ para los casos $n=5,10,20$, en el caso de que los nodos sean igualmente espaciados y en el caso que sean nodos de TChebyshev. En cada caso usar un mismo sistema de coordenadas.



4 ECUACIONES NO LINEALES.

En general, no es posible determinar los *ceros* de una función, es decir, valores x^* tal que $f(x^*) = 0$, en un número finito de pasos. Tenemos que usar métodos de aproximación. Los métodos son usualmente iterativos y tienen la forma: Iniciando con una aproximación inicial x_0 (o un intervalo $[a,b]$), se calculan aproximaciones sucesivas $x_1, x_2, ...$ y elegimos x_n como aproximación de x^* cuando se cumpla un *criterio de parada dado.* A los ceros de un polinomio se les conoce también como *raíces.*

En este capítulo veremos los métodos iterativos usuales: bisección, regula falsi, punto fijo, Newton y el método de la secante. Además se incluyen los teoremas de convergencia el análisis del orden de convergencia y, en algunos casos, análisis de las cotas de error. En todo caso, se incluye una sección para establecer el criterio de parada de un algoritmo. En general, no usamos estos métodos de manera aislada sino más bien combinada. Por eso se incluyen secciones con métodos híbridos. El método de bisección es muy confiable, pero relativamente lento. Los métodos de Newton y la secante son más veloces, pero no son tan confiables como bisección. Bisección es óptimo para funciones continuas (en general) pero no para funciones derivables o convexas. En este último caso, el método de Newton es veloz, pero necesita el cálculo de la derivada (no todas las funciones derivables tienen derivadas que se pueden expresar en términos de funciones elementales o funciones especiales) y podría colapsar si la derivada toma valores muy pequeños en el proceso. En las funciones obtenidas por interpolación corre el riesgo de caer en un ciclo. El método de la secante es veloz y no requiere la derivada, sino una aproximación. Aún así, corre riesgos inherentes al comportamiento de la derivada en las cercanías de la raíz.

El método de Dekker-Brent combina la confiabilidad de bisección con la velocidad del método de la secante y el método de interpolación cuadrática inversa. Iniciando con un intervalo donde la función cambia de signo, el siguiente paso toma el camino más veloz disponible (secante o interpolación cuadrática inversa, si no hay peligro de colapso) sin dejar nunca el intervalo donde hay cambio de signo. Así en el peor de los casos, en el siguiente paso se usaría bisección. Este algoritmo es un método de tipo adaptativo. Está balanceado de tal manera que siempre encuentra una respuesta y es siempre más rápido que bisección. Algunos paquetes de software, como *Mathematica*® y *MatLab*®, usan Newton, secante y el método de Brent para aproximar ceros de funciones. Para encontrar las raíces de un polinomio se usan algoritmos especializados, por ejemplo el algoritmo de Jenkins-Traub.

4.1 Orden de convergencia

El orden de convergencia nos da una 'medida' de la rapidez con la que una sucesión de aproximación converge, en particular nos podría informar con qué rapidez, a partir de cierto índice, ganamos cifras decimales correctas.

Definición 4.1 (Orden de convergencia).

Supongamos que $\lim_{n\to\infty} u_n = u^*$ y que $u_n \neq u^*$ $\forall n$. Se dice que la sucesión tiene *orden de convergencia* $q \geq 1$ si

$$\lim_{n\to\infty} \frac{|u_{n+1} - u^*|}{|u_n - u^*|^q} = K \quad \text{para alguna constante} \quad 0 < K < \infty. \tag{4.1}$$

Para entender lo que dice la definición, supongamos que δ_n denota el número de lugares decimales exactos en la aproximación en la $n-$ésima iteración, entonces

$$\delta_n \approx -\log_{10}|u_n - u^*|$$

Si $u_n = 0.123447$ y $u^* = 0.123457$ entonces $\delta_n \approx -\log_{10}|0.00001| = 5$

Si $u_n = 1.53222$ y $u^* = 1.5332423$ entonces $\delta_n \approx -\log_{10}|0.00001| = 2.99042$

Tomando logaritmos de base 10 a ambos lados del límite (4.1) podemos establecer que

$$\delta_{n+1} \approx q\,\delta_n - \log_{10}|K|.$$

Es decir, si el orden de convergencia es q, en la siguiente iteración (después de la iteración $n-$ésima), el número de lugares decimales exactos es aproximadamente $q\,\delta_n + L$ con $L = -\log_{10}|K|$.

- q no necesariamente es un entero.
- Si $q = K = 1$, u_n converge más despacio que una sucesión que converja linealmente y se dice que converge *sublinealmente.*
- Si (4.1) se da con $q = 1$ y $K = 0$, pero no con $q > 1$, la convergencia se dice *superlineal.*
- Si $q = 1$ y $0 < K < 1$, u_n se dice que *converge linealmente* y K es la *tasa* de convergencia.
- Si $q = 2$ entonces u_n se dice que *converge cuadráticamente* y se espera que a partir de algún subíndice N, cada iteración aproximadamente *duplica* el número de lugares decimales exactos. Observe que en el caso particular $q = 2$ y $K = 1$,

$$\begin{aligned} \delta_{n+1} &\approx q\,\delta_n - \log_{10}|K| \\ &\approx 2\,\delta_n. \end{aligned}$$

Ejemplo 4.1

$q=1$ indica que se gana una cifra decimal exacta cada cierto número promedio de iteraciones. La sucesión $x_n = \frac{1}{6}-\frac{1}{3}\left(\frac{-1}{2}\right)^n$ converge a $1/6 = 0.166666666...$

El orden de convergencia es $q=1$. En efecto,

$$\lim_{n\to\infty}\frac{|u_{n+1}-1/6|}{|u_n-1/6|^q} = \lim_{n\to\infty}\frac{\frac{1}{3}\left(\frac{1}{2}\right)^{n+1}}{\left(\frac{1}{3}\right)^q\left(\frac{1}{2}\right)^{nq}} = \lim_{n\to\infty} 3^{1-q}\left(\frac{1}{2}\right)^{n(1-q)+1} = \begin{cases} \frac{1}{2} & \text{si} \quad q=1 \\ \infty & \text{si} \quad q>1. \end{cases}$$

En la tabla se observa que efectivamente $\delta_{n+1} \approx \delta_n$ y hay que esperar algunas iteraciones adicionales para ganar un nuevo decimal exacto.

n	δ_{n+1}	δ_n	x_n
40	12.8193775849834	12.5183078278654	0.1666666666666818
	13.1203280613792	12.8193775849834	0.1666666666666591
	13.4215171101422	13.1203280613792	0.1666666666666705
	13.7222290578374	13.4215171101422	0.1666666666666648
	14.0238953825265	13.7222290578374	0.1666666666666676
	14.3236536511268	14.0238953825265	0.1666666666666662
	14.6272308358047	14.3236536511268	0.1666666666666669
	14.9231813059394	14.6272308358047	0.1666666666666665
	15.2344304667850	14.9231813059394	0.1666666666666667
49	15.5152570763607	15.2344304667850	0.1666666666666666

Ejemplo 4.2 (Convergencia cuadrática).

Sea $x_0>0$ y $x_{n+1}=0.5\left(x_n+\frac{2}{x_n}\right)$. Se sabe que $\lim_{n\to\infty} x_n = \sqrt{2}$. En este caso, la convergencia es cuadrática: En efecto, primero veamos que $x_{n+1}-\sqrt{2} = \left(x_n-\sqrt{2}\right)^2$, entonces

$$\frac{x_{n+1}-\sqrt{2}}{(x_n-\sqrt{2})^q} = \frac{\left(x_n-\sqrt{2}\right)^2}{2x_n\left(x_n-\sqrt{2}\right)^q} = \frac{\left(x_n-\sqrt{2}\right)^{2-q}}{2x_n}$$

por tanto,

$$\lim_{n\to\infty}\left|\frac{\left(x_n-\sqrt{2}\right)^{2-q}}{2x_n}\right| = \begin{cases} 0 & \text{si} \quad q=1 \\ \frac{1}{2\sqrt{2}} & \text{si} \quad q=2 \\ \infty & \text{si} \quad q>2 \end{cases}$$

Ejemplo 4.2 (continuación).

Comparemos $\sqrt{2} = 1.4142135623730950...$ con los valores de la segunda columna de la tabla (4.2).

n	x_n	$\left\lvert x_n - \sqrt{2} \right\rvert$
0	1.5	0.5
1	1.41666666666667	0.083333333
2	1.41421568627451	0.00245098
3	1.41421356237469	2.1239×10^{-6}
4	1.41421356237310	1.59472×10^{-12}
5	1.41421356237309	2.22045×10^{-16}

La convergencia es muy rápida y se puede observar como *aproximadamente,* la precisión se duplica (en las cercanías de $\sqrt{2}$) en cada iteración.

EJERCICIOS

4.1 La sucesión $x_{n+1} = \dfrac{2x_n^3 - 2}{3x_n^2 - 3}$ converge a -2 iniciando en $x_0 = -2.4$.

a) Verifique que la convergencia es cuadrática.

4.2 Sean $x_0 > 0,\ A \geq 0$ y $x_{n+1} = 0.5\left(x_n + \dfrac{A}{x_n}\right)$. Se sabe que $\lim\limits_{n \to \infty} x_n = \sqrt{A}.$ Verifique que la convergencia es cuadrática.

Aunque en general no disponemos de un mecanismo analítico para resolver una ecuación arbitraria, si tenemos métodos para aproximar una solución mediante aproximaciones sucesivas. En lo que resta del capítulo vamos ver algunas de estos métodos, su confiabilidad y su velocidad de convergencia.

Vamos a establecer algunos detalles acerca de notación que usaremos en este capítulo.

- $f \in C^n[a,b]$ indica que f tiene derivadas continuas hasta el orden n en $[a,b]$. En particular, $f \in C[a,b]$ indica que f es continua en $[a,b]$ y $f \in C^1[a,b]$ indica que f es derivable en $[a,b]$
- Recordemos los puntos críticos en $[a,b]$ de $f \in C^1[a,b]$ son las soluciones de la ecuación $f'(x) = 0$ en $[a,b]$. Si los puntos críticos en $[a,b]$ son $\{x_0, x_1, ..., x_k\}$ entonces el *máximo* y el *mínimo* valor del conjunto

$$\{f(a), f(b), f(x_0), f(x_1), ..., f(x_k)\}$$

  corresponden al *máximo y el mínimo absoluto* de esta función en $[a,b]$.
- Si f es una función de una variable y si $f(x^*) = 0$ entonces decimos que x^* es un *cero* de f. Si f es un polinomio también se dice que x^* es una *raíz.*

4.2 Método de Punto Fijo

En muchos casos una ecuación no lineal aparece en la forma de "*problema de punto fijo*":

$$\text{Encuentre } x \text{ tal que } \; x = g(x) \tag{4.2}$$

Un número $x = x^*$ que satisface esta ecuación se llama *punto fijo de* g.

Ejemplo 4.3

- En el problema de punto fijo $x = \operatorname{sen}(x)$, tenemos $g(x) = \operatorname{sen}(x)$ y un punto fijo de g es $x^* = 0$.
- Si $g(x) = x^2$, los puntos fijos son $x = 0$ y $x = 1$ pues $0^2 = 0$ y $1^2 = 1$.
- Si $g(x) = \ln x + x$, un punto fijo es $x = 1$ pues $\ln 1 + 1 = 1$.

¿Cuáles son los puntos fijos de $g(x) = \ln(x)$ o de $g(x) = \cos(x)$?. Geométricamente, un punto fijo corresponde al valor de la abscisa donde la gráfica de $y = g(x)$ interseca a la recta $y = x$.

Ejemplo 4.4

En el problema de punto fijo $x = \cos(x)$, tenemos $g(x) = \cos(x)$. El único punto fijo de g es $x^* \approx 0.7390851332151607...$

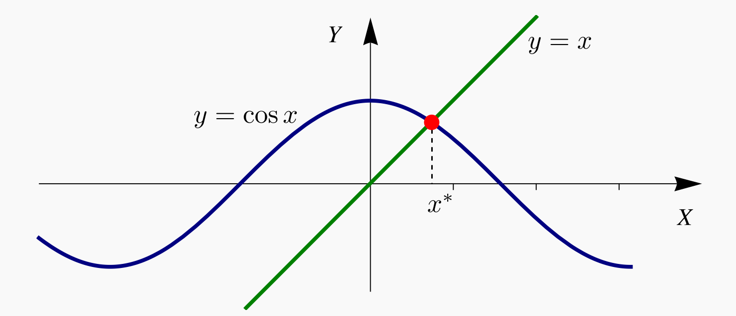


Figura 4.1 Punto fijo de $\cos(x)$: $x^* \approx 0.73908...$

Una ecuación $f(x) = 0$ se puede escribir en la forma (4.2) despejando x (si se pudiera). En este caso se obtiene $f(x) = 0 \implies x = g(x)$

Ejemplo 4.5

La ecuación $x^3 + x + 1 = 0$ se puede poner como un problema de punto fijo despejando x de varias maneras,

a.) $x = -x^3 - 1$. En este caso $g(x) = -x^3 - 1$

b.) $x = \sqrt[3]{-x-1}$. En este caso $g(x) = \sqrt[3]{-x-1}$

c.) $x = \dfrac{\sqrt{-1-x}}{x^2}$

d.) ...

Iteración de punto fijo. Un esquema iterativo de punto fijo define por recurrencia una sucesión $\{x_n\}$ de la siguiente manera:



$$x_0 \in \mathbb{R}, \;\; x_{i+1} = g(x_i), \; i = 0, 2, ... \qquad (4.3)$$

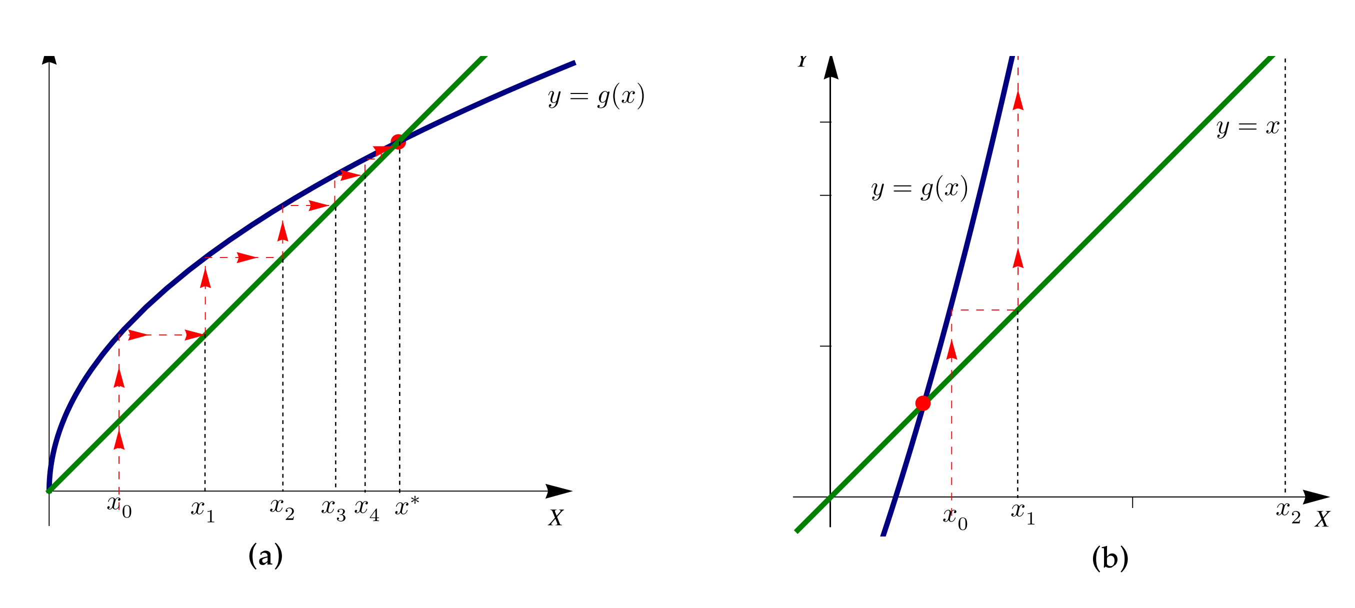


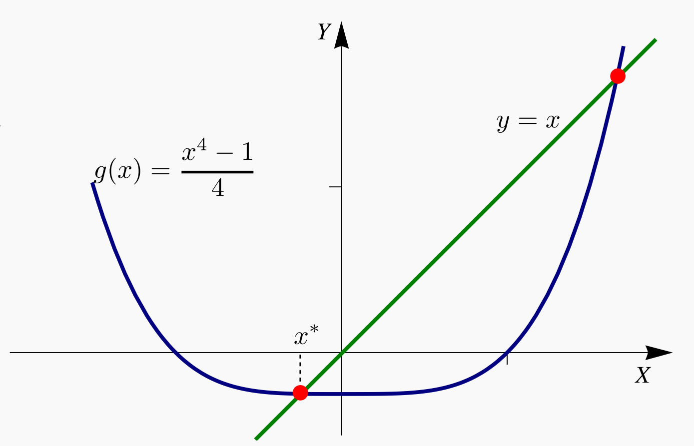
Figura 4.2 Iteraciones del método de punto fijo: convergencia y divergencia. Observe que $|g'(x)| < 1$ en los alrededores de $x = x^*$ en la figura **(a)**

Teorema 4.1

Sea la sucesión $\{x_n\}$ definida por $x_0 \in \mathbb{R}, \;\; x_{i+1} = g(x_i), \; i = 0, 2, ...$ con g continua. Si $\lim\limits_{\to\infty} x_n = x^*$, entonces x^* es punto fijo de g.

Ejemplo 4.6

Consideremos la ecuación $x = \frac{x^4 - 1}{4}$, y sea $x_0 = 0.2$, la iteración de fijo correspondiente es



$x_0 = 0.2, \quad g(x) = (x^4 - 1)/4$
$x_1 = g(x_0) \approx -0.2496$
$x_2 = g(x_1) \approx -0.2490296...$
$x_3 = g(x_2) \approx -0.2490385...$
$\ldots$

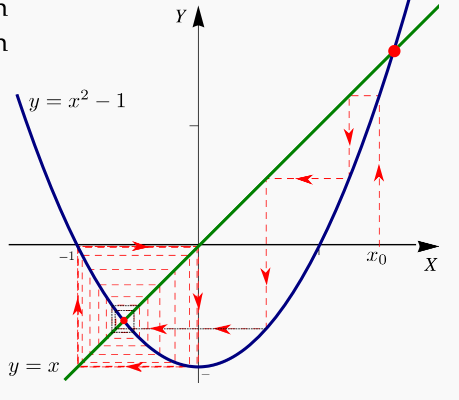
Toda la información aparece en la tabla

n	x_{n+1}	Error absoluto estimado: $\lvert x_{n+1} - x_n \rvert$	Error relativo estimado: $\lvert (x_{n+1} - x_n)/x_{n+1} \rvert$
1	-0.2496	0.4496	1.801282051
2	-0.249029672515994	0.000570327	0.002290199
3	-0.249038510826169	8.83831×10^{-6}	3.54897×10^{-5}
4	-0.249038374322083	1.36504×10^{-7}	5.48125×10^{-7}
5	-0.249038376430443	2.10836×10^{-9}	8.46601×10^{-9}
6	-0.249038376397879	3.25645×10^{-11}	1.30761×10^{-10}
7	-0.249038376398382	5.02959×10^{-13}	2.0196×10^{-12}

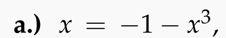
Ejemplo 4.7

Consideremos el problema de punto fijo $x^2 - 1 = x$. De acuerdo a la figura, hay un punto fijo en $[1,2]$ y otro en $[-1,0]$ Si tomamos $x_0 = 1.5$, la iteración de punto fijo parece divergente según los datos de la tabla de la derecha. En este caso, la iteración cae en un ciclo si para algún n, $x_n = 0$ o $x_n = -1$.

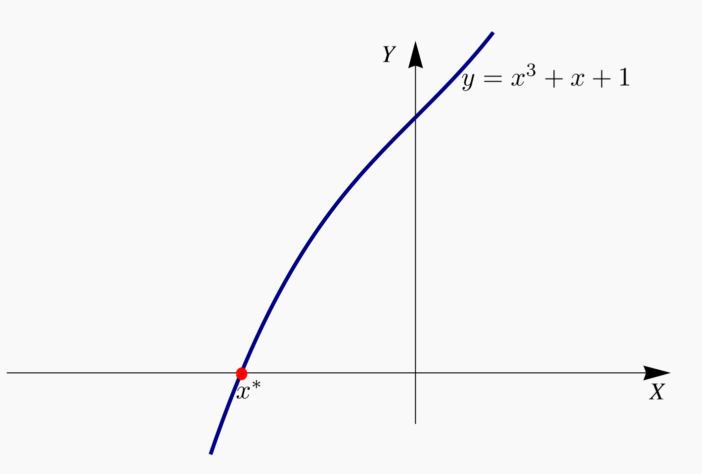
n	x_{n+1}	Error estimado
1	1.25000000	0.25
2	0.56250000	0.6875
3	-0.68359375	1.2460937
...		...
22	0	1
23	-1	1
24	0	1
25	-1	1
26	0	1



En ambos casos hay un entorno alrededor de x^* en el que $|g'(x)| \not< 1$.



Aunque no es un método eficiente (en esta forma cruda), podemos aproximar la única solución real de $x^3 + x + 1 = 0$ usando iteración de punto fijo.
Para empezar, debemos poner el problema como un problema de punto fijo. Para hacer esto debemos despejar "x". Hay varias posibilidades.



a.) $x = -1 - x^3$,

b.) $x = \sqrt[3]{-1-x}$,

c.) $x = \dfrac{-1-x}{x^2}$,

d.) ...

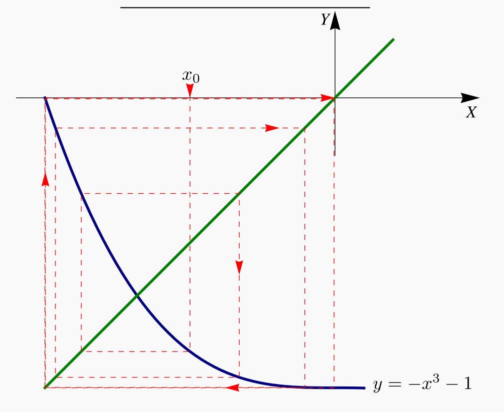
El resultado de aplicar punto fijo en los dos primeros casos, con $x_0 = -0.5$, se puede ver en las tablas que siguen.

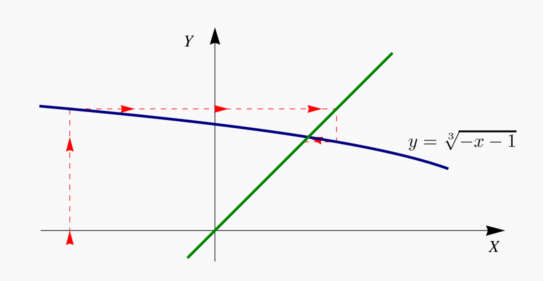
$x = -1 - x^3$

n	x_{n+1}
1	-0.7840000000
2	-0.5181096960
3	-0.8609198471
4	-0.3619008595
5	-0.9526010366
...	...
17	-1
18	0
19	-1
20	0
21	-1
22	0
...	

$x = \sqrt[3]{-x-1}$, $|g'(x)| < 1$

n	x_{n+1}	$\lvert x_n - x_{n+1}\rvert$
1	-0.79370052	0.293700526
2	-0.59088011	0.202820413
3	-0.74236393	0.151483819
4	-0.63631020	0.106053729
...		
24	-0.6822715	0.000134782
25	-0.6823680	9.65013×10^{-5}
26	-0.6822989	6.90905×10^{-5}
27	-0.6823484	4.94671×10^{-5}
28	-0.6823130	3.54165×10^{-5}
29	-0.6823383	2.53572×10^{-5}
30	-0.6823202	1.81548×10^{-5}





Los teoremas que siguen dan condiciones *suficientes* (pero no necesarias) para que haya un punto fijo en un intervalo, condiciones suficientes para que el punto fijo sea único en el intervalo y condiciones suficientes para la convergencia.

Teorema 4.2

Si $g \in C[a,b]$ y si $g(x) \in [a,b]$ para todo $x \in [a,b]$, entonces g tiene un punto fijo en $[a,b]$

Teorema 4.3

Si $g \in C[a,b]$ y si $g(x) \in [a,b]$ para todo $x \in [a,b]$ y si g' esta definida en $]a,b[$ y cumple $|g'(x)| < 1$ en este intervalo, entonces el punto fijo es único.

Teorema 4.4

Si $g \in C[a,b]$ y si $g(x) \in [a,b]$ para todo $x \in [a,b]$ y si g' esta definida en $]a,b[$ y existe k positiva tal que $|g'(x)| \leq k < 1$ en $]a,b[$, entonces, para cualquier $x_0 \in [a,b]$, la iteración $x_{n+1} = g(x_n)$ converge a un único punto fijo x^* de g en este intervalo. También

$$|x_{n+1} - x^*| \leq \frac{k^n |x_1 - x_0|}{1-k}, \ \ n = 1,2,...$$

El orden de convergencia de este método coincide con la multiplicidad del punto fijo, es decir, si $g^{(p)}$ es la primera derivada que no se anula en x^* entonces el orden de convergencia es p.

Ejemplo 4.9

Sea $g(x) = (x^3 - 1)/4$. En el intervalo $[-1,1]$, esta función tiene un punto fijo único x^* y además $\lim_{n\to\infty} g(x_n) = x^*$.

Para mostrar esta afirmación verifiquemos las condiciones de la parte *c.)* del teorema (4.5).

- g es continua y $g(x) \in [-1,1]$ para todo $x \in [-1,1]$

  Efectivamente, g es continua y como $g'(x) = \dfrac{3x^2}{4}$ entonces g solo tiene un punto crítico en $[-1,1]$, $x = 0$. Luego,

$$\text{Mín}\{g(-1), g(1), g(0)\} \leq g(x) \leq \text{Máx}\{\{g(-1), g(1), g(0)\}$$

Ejemplo 4.9 (continuación).

$$\text{Mín}\{-\frac{1}{2},0,-\frac{1}{4}\} \leq g(x) \leq \text{Máx}\{-\frac{1}{2},0,-\frac{1}{4}\} \implies \mathbf{-1} \leq -\frac{1}{2} \leq g(x) \leq 0 \leq \mathbf{1}$$

por lo tanto, $g(x) \in [-1,1]$. Con lo cual g tiene un punto fijo en $[-1,1]$.

- Si g' existe en $]-1,1[$ y $|g'(x)| \leq k < 1$ para toda $x \in]-1,1[$ entonces el punto fijo es único y la iteración de punto fijo converge para cualquier $x_0 \in [-1,1]$.

Como $g'(x) = \dfrac{3x^2}{4}$ entonces $g''(x) = \dfrac{3x}{2}$ tiene un punto crítico en el intervalo, a saber $x = 0$. Por lo tanto

$$|g'(x)| \leq \text{Máx}\{|g'(-1)|,|g'(1)|,|g(0)|\} \implies |g'(x)| \leq \frac{3}{4} < \mathbf{1} \text{ si } x \in [-1,1]$$

por lo que, en particular

$$|g'(x)| \leq \frac{3}{4} < 1 \text{ en }]-1,1[$$

y podemos tomar $k = 3/4$. Por lo tanto, para cualquier $x_0 \in [-1,1]$, $\lim\limits_{n\to\infty} g(x_n) = x^*$.

Además, si $x_0 = -0.5$, *una* cota de error en la $n-$ésima iteración de punto fijo se puede establecer como

$$|x_n - x^*| \leq \frac{(3/4)^n}{1-3/4}|-0.5-g(-0.5)| \implies |x_n - x^*| \leq 3^n 4^{1-n} \frac{7}{32}$$

En particular, cuando $n = 5$, $x_5 = -0.2541018552...$ y entonces $|x_5 - x^*| \leq 0.2076416015625$, mientras que la estimación del valor absoluto, más apegada a la realidad en este caso, nos da $3{,}27797 \times 10^{-6}$

4.3 Punto Fijo. Algoritmo e Implementación.

El algoritmo es muy sencillo. Para la implementación del método de punto fijo, el criterio de parada que podríamos usar es

$$|x_n - x_{n-1}| \leq \delta(|x_n| + 1) \text{ y un número máximo de iteraciones.}$$

Esta es una combinación entre error relativo y error absoluto con un número máximo de iteraciones.

Algoritmo 4.1: Iteración de Punto fijo.

Datos: Una función continua g, x_0, δ, maxItr.
Salida: Si hay convergencia, una aproximación x_n de un punto fijo.

```
k = 0;
repeat
    x1 = g(x0);
    dx = |x1 − x0|;
    x0 = x1;
    k = k + 1;
until dx ≤ δ(|x0| + 1) o k > maxItr ;
return x1
```

Implementación Basic. La implementación en Basic del método de punto fijo los valores x_0, δ y maxItr. La función $g(x)$ se define en el código (`Alt-F11`).

Código VBA 4.1: Función `PuntoFijo`

```
Function gx(x)
    gx= -1-x^3
End Function
'-------------------------------------------------------
Function PuntoFijo(xc, delta, maxItr)
 Dim k, x0, x1, dx
 k = 0
 x0 = xc
 Do
    x1 = g.Eval1(x0)
    dx = Abs(x0 - x1)
    x0 = x1
    k = k + 1
 Loop Until dx <= delta Or k > maxItr
 PuntoFijo = x1
End Function
```

Implementación con XNumbers. Esta segunda implementación usa la extensión Xnumbers para leer y evaluar la función g. La función `PuntoFijo` recibe la función g como "string", y los valores x_0, δ y maxItr.

Actualmente hay XNumbers en versión 'Excel 97/2000/XP/2003' y versión 'Excel2010and2007'. Las instrucciones (también puede ver el apéndice sobre programación VBA Excel) y el complemento se puede descargar (agosto 2012) desde `http://www.thetropicalevents.com/Xnumbers60/`. La fuente original de Xnumbers (para Excel 97/2000/XP/2003) es `http://digilander.libero.it/foxes/`.

Código VBA 4.2: Función `PuntoFijo` con la extensión `clsMathParser`

```
Function PuntoFijo(gx, xc, delta, maxItr)
Dim g As New clsMathParser   'Crea un objeto clsMathParser 'g'. Desde g podemos acceder a todos
                             'los métodos, en particular g.Eval1(x) para evaluar en x
Dim k, x0, x1, dx, okgx

okgx = g.StoreExpression(gx) 'lee la función g(x)=gx (un string), revisa sintaxis y almacena.
                             'Devuelve true o false
If Not okgx Then
    MsgBox ("Error en g: " + g.ErrorDescription) 'si hubiera error....
    Exit Function
End If
k = 0
x0 = xc
Do
    x1 = g.Eval1(x0)
    dx = Abs(x0 - x1)
    x0 = x1
    k = k + 1
Loop Until dx <= delta Or k > maxItr
PuntoFijo = x1
End Function
```

Hoja Excel para el método de punto fijo. Para hacer una hoja Excel, modificamos la función "PuntoFijo" y la convertimos en una subrutina para poder imprimir en las celdas. Como referencia tenemos la figura (4.3). Observe que en este cuaderno se supone que hemos instalado la extensión XNumbers, por eso podemos leer la función desde la celda 'D3'.

	A	B	C	D	E	F	G	H	I
1	Este cuaderno usa 'XNUMBERS'								
2	**Iteración Punto Fijo: Aproxima una solución del problema g(x) = x**								
3			g(x) =	root(-x - 1,3)					
4				APROXIMACION	Error Estimado		Ejecutar Iteración Punto Fijo		
5	N	TOL	X_0	$X_N = g(X_{N-1})$	$\|x_{n-1} - x_{n-2}\|$	$g(x_n)$			
6	4	0,01	-0,6	-0,7368063	0,1368063	-0,64085311			
7				-0,64085311	9,60E-02	-0,71081629			
8				-0,71081629	7,00E-02	-0,66128897			
9				-0,66128897	4,95E-02	-0,69707009			
10									

Figura 4.3 Iteración de punto fijo.

 **Software:** Cuadernos LibreOffice y Excel

Código VBA 4.3: Subrutina que se activa con el botón en la hoja Excel.

```
Private Sub CommandButton1_Click()
Dim gx, xc, delta, maxItr
    gx = Cells(3, 2)
    xc = Cells(8, 1)
```

```
    delta = Cells(8, 2)
    maxItr = Cells(8, 3)
    Call PuntoFijo(gx, xc, delta, maxItr, 8, 5)
End Sub
Sub PuntoFijo(gx, xc, delta, maxItr, fi, co)
Dim g As New clsMathParser
Dim k, x0, x1, dx, okgx
okgx = g.StoreExpression(gx)
If Not okgx Then
    MsgBox ("Error en g: " + g.ErrorDescription)
    Exit Sub
End If
k = 0
x0 = xc
Do
    x1 = g.Eval1(x0)
    dx = Abs(x0 - x1)
    x0 = x1
    Cells(fi + k, co) = x1
    Cells(fi + k, co + 1) = dx
k = k + 1
Loop Until dx <= delta Or k > maxItr
End Sub
```

EJERCICIOS

4.3 Usar punto fijo para aproximar la solución de cada ecuación en el intervalo que se indica.

a) $x^3 - x - 1 = 0$ en $[1,2]$
b) $x^2 - x - 1 = 0$ en $[-1,0]$
c) $x = e^{-x}$ en $[0,1]$
d) $x^5 - x + 1 = 0$ en $[-2,-1]$
e) $\text{sen}(x) - x = 0$ en $[-1,1]$

4.4 Verifique que las siguientes funciones tienen un único punto fijo en el intervalo dado.
a) $g(x) = (x^2 - 1)/3$ en el intervalo $[-1,1]$.
b) $g(x) = 2^{-x}$ en $[1/3,1]$

4.5 Repita la parte 1.) del ejemplo (4.9) con $g(x) = (x^3 - 1)/4$ en $[-0.5,0]$.

4.6 Considere $g(x) = 2^{-x}$.
a) ¿Podría encontrar una constante positiva $k < 1$ tal que $|g'(x)| \leq k \ \forall x \in]1/3,1[$?
b) ¿Se puede garantizar que la iteración de punto fijo, iniciando en cualquier $x_0 \in [1/3,1]$, converge al único punto fijo de g en el intervalo $[1/3,1]$?.

4.7 Considere $x = 0.5(\text{sen}(x) + \cos(x))$. Determine un intervalo $[a,b]$ dónde la iteración de punto fijo converge sin importar la elección de la aproximación inicial $x_0 \in [a,b]$. Debe justificar su respuesta.

4.8 Considere el problema de punto fijo: $x = 0.5(x + 2/x)$. Determine un intervalo [a,b] en el que se pueda aplicar la parte $c.)$ del teorema (4.5). **Nota:** Es claro que $x = 0.5(x + 2/x)$ tiene como solución $\pm\sqrt{2}$. La iteración solo se usa con el propósito de obtener una aproximación "tangible" de esta raíz, con algunos decimales exactos.

4.9 Usando la implementación en VBA Excel, aplique iteración de punto fijo para resolver el problema $x = (2x^3 - 2)/(3x^2 - 3$ tomando $x_0 = 1.2$. Hay algo muy extraño pasando aquí. ¿Qué es?

4.10 Considere el problema de punto fijo $x = e^{-x}$. Muestre que la iteración de punto fijo converge para cualquier $x_0 > 0$.

4.11 Considere el problema de punto fijo $x = 3x - 3x^2 + x^3$.

a) Determine un intervalo $[a,b]$ tal que la iteración de punto fijo converja a $x^* = 1$, para todo $x_0 \in [a,b]$. **Ayuda:** Haga una representación gráfica de g, g' y 1.

4.4 Punto Fijo: Aspectos Teóricos.

El teorema que sigue establece condiciones suficientes para la existencia de un punto fijo en un intervalo y también condiciones para la convergencia del proceso de iteración de punto fijo.

Teorema 4.5 (Punto fijo).

Sea $g \in C[a,b]$,

a.) Si $g(x) \in [a,b]$ para todo $x \in [a,b]$, entonces g tiene un punto fijo en $[a,b]$

b.) Si g' esta definida en $]a,b[$ y si $|g'(x)| < 1$ en este intervalo, entonces el punto fijo es único.

c.) Si $g' \in C[a,b]$ y si $g(x) \in [a,b]$ para todo $x \in [a,b]$ y además existe k positiva tal que $|g'(x)| \leq k < 1$ en $]a,b[$, entonces, para cualquier $x_0 \in [a,b]$, la iteración $x_{n+1} = g(x_n)$ converge a un único punto fijo x^* de g en este intervalo. También

$$|x_{n+1} - x^*| \leq \frac{k^n |x_1 - x_0|}{1 - k}, \quad n = 1,2,...$$

Prueba

a.) La idea es tomar $f(x) = x - g(x)$ y mostrar que cambia de signo, así habría un cero x^* de f, es decir, $x^* - g(x^*) = 0$.

Sea $g(a) \neq a$ o $g(b) \neq b$ (sino, el teorema se cumple directamente). En este caso, por hipótesis, $a < g(x) < b$ y entonces la función $f(x) = x - g(x)$ cambia de signo en $[a,b]$ pues $f(a) < 0$ y $f(b) > 0$.

como f es continua en $[a,b]$ entonces f se anula en este intervalo, es decir existe $x^* \in]a,b[$ tal que $f(x^*) = 0$ o $g(x^*) = x^*$.

b.) La idea es esta: si $|g'(x)| \not< 1$ cabe la posibilidad de que g vuelva sobre la recta $y=x$, en otro caso, esto no sería posible (figura 4.4). Si hay varios puntos fijos tendríamos una situación como la de la figura (4.5) y en este caso, por el teorema del valor medio, existe un $\xi \in]a,b[$ tal que $|g'(\xi)| = 1$ y esto contradice la hipótesis $|g'(x)| < 1$.

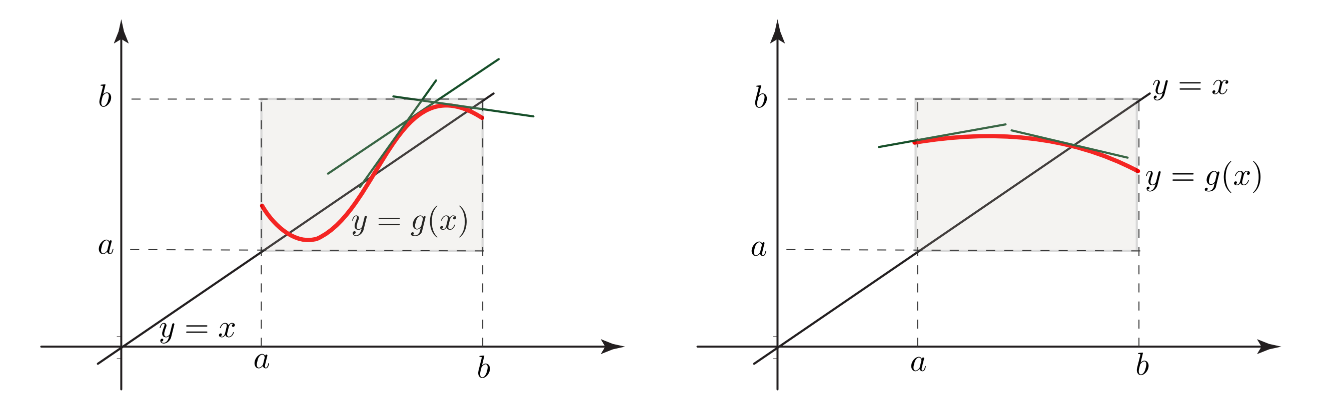


Figura 4.4 Si $x \in]a,b[$, en *a.)* $|g'(x)| \not< 1$, y en *b.)* $|g'(x)| < 1$

Par mostrar la unicidad del punto fijo procedemos por contradicción. Supongamos que tenemos dos puntos fijos distintos en $[a,b]$, x_1^* y x_2^*. Por el teorema del valor medio, existe $\xi \in]a,b[$ tal que

$$g'(\xi) = \frac{g(x_1^*) - g(x_2^*)}{x_1^* - x_2^*}$$

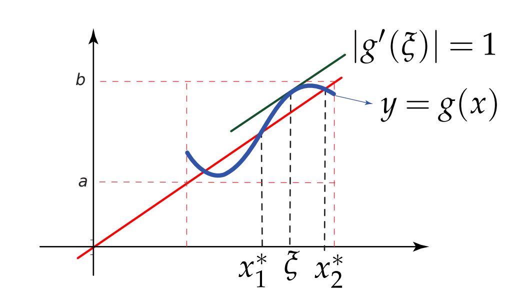


Figura 4.5 Teorema del valor medio

Como asumimos que $g(x_1^*) = x_1^*$ y $g(x_2^*) = x_2^*$ entonces

$$g'(\xi) = \frac{x_1^* - x_2^*}{x_1^* - x_2^*} = 1$$

y esto contradice la hipótesis $|g'(x)| < 1$ en $]a,b[$.

c.) La idea aquí es usar la hipótesis y el teorema del valor medio para establecer la desigualdad $|x_n - x_{n+1}| \leq k^n|x_0 - x_1|$. Aplicando esta desigualdad a $x_n, x_{n+1}, ..., x_{n+m-n}$ llegamos a la desigualdad

$$|x_m - x_n| \leq |x_0 - x_1||k^n(1 + k + k^2 ... + k^{m-n-1})|$$

y concluimos el resultado tomando límites a ambos lados con $m \longrightarrow \infty$.

Según las hipótesis y de acuerdo a los enunciados en *a.)* y *b.)* ya sabemos que g tiene un único punto fijo x^* en $[a,b]$. Primero vamos a mostrar que $\lim_{x\to\infty} g(x_n) = x^*$ con $x_0 \in [a,b]$ arbitrario.

Sabemos que $x_0 \in [a,b]$ y $x_n = g(x_{n-1}) \in [a,b]$, $n = 1,2,...$ Ahora, de acuerdo al teorema del valor medio, existe $\xi_n \in]a,b[$ tal que

$$|x^* - x_{n+1}| = |g(x^*) - g(x_n)| \leq |g'(\xi_n)||x^* - x_n|,\ n = 0,1,...$$

Como $|g'(x)| \leq k < 1$ en $]a,b[$, entonces

$$|g(x^*) - g(x_n)| \leq k|x^* - x_n|,\ n = 0,1,...$$

Luego

$$|x^* - x_{n+1}| \leq k|x^* - x_n| \leq k^2|x^* - x_{n-1}| \leq k^3|x^* - x_{n-2}| \leq ... \leq k^{n+1}|x^* - x_0|$$

por lo que

$$|x^* - x_{n+1}| \leq k^{n+1}|x^* - x_0|$$

Finalmente, como $0 < k < 1$ entonces usando el teorema de intercalación llegamos a que

$$\lim_{x \to \infty} x_n = x^*.$$

Para mostrar la última afirmación del teorema aplicamos el razonamiento anterior con $|x_{n+1} - x_n|$ en vez de $|x^* - x_{n+1}|$. Por el teorema del valor medio, existe $\xi_n \in]a,b[$ tal que

$$|x_n - x_{n+1}| = |g(x_{n-1}) - g(x_n)| \leq |g'(\xi_n)||x_{n-1} - x_n|,\ n = 1,2,...$$

por lo que

$$|x_n - x_{n+1}| \leq k|x_{n-1} - x_n| \leq k^2|x_{n-2} - x_{n-1}| \leq k^3|x_{n-3} - x_{n-2}| \leq ... \leq k^n|x_0 - x_1|$$

es decir

$$|x_n - x_{n+1}| \leq k^n|x_0 - x_1| \tag{4.4}$$

Luego, si $m > n \geq 1$ (usaremos los números $n, n+1, n+2, ..., n+m-n$)

$$\begin{aligned}
|x_m - x_n| &= |x_m - x_{m-1} + x_{m-1} - x_{m-2} + x_{m-2} + ... + x_{n+1} - x_n| \\
&\leq |x_m - x_{m-1}| + |x_{m-1} - x_{m-2}| + ... + |x_{n+1} - x_n| \\
&\leq k^{m-1}|x_0 - x_1| + k^{m-2}|x_0 - x_1| + ... + k^n|x_0 - x_1|, \text{ por (4.4)} \\
&\leq |x_0 - x_1||k^{m-1} + k^{m-2} + ... + k^n| = |x_0 - x_1||k^n + k^{n+1} + ... + k^{m-1}| \\
&\leq |x_0 - x_1||k^n(1 + k + k^2 ... + k^{m-n-1})|
\end{aligned}$$

Ahora si tomamos n fijo (x_n sería una constante) tomando límites a ambos lados de la desigualdad y haciendo $m \to \infty$

$$|x^* - x_n| = \lim_{m \longrightarrow \infty} |x_m - x_n| \leq |x_0 - x_1| k^n \sum_{i=1}^{\infty} k^i = \frac{k^n}{1-k}|x_0 - x_1|$$

con lo que $|x^* - x_n| \leq \dfrac{k^n}{1-k}|x_0 - x_1|$. Note que usamos la fórmula (serie geométrica) $1 + r + r^2 + \cdots = \dfrac{1}{1-r}$, $|r| < 1$.

La pregunta natural es: ¿Si $|g'(x)| > 1$ en $]a,b[$ entonces hay divergencia?

Si $g(x^*) = x^*$ y si $x_{n+1} = g(x_n)$ entonces restando x^* a ambos lados tenemos

$$\begin{aligned}
x_{n+1} - x^* &= g(x_n) - g(x^*) \\
&= g(x_n) - g(x_n) + g'(\xi_n)(x_n - x^*), \ \xi_n \text{ entre } x^* \text{ y } x_n \\
&= g'(\xi_n)(x_n - x^*) \\
\therefore \ |x_{n+1} - x^*| &= |g'(\xi_n)||(x_n - x^*)|.
\end{aligned}$$

Si g' es continua, $|g'(\xi_n)| \longrightarrow |g'(x^*)|$ y entonces no habría convergencia si $|g'(x^*)| > 1$, excepto que algún $x_i = x^*$.

Sea $m > 1$ y $g^{(j)}(x^*) = 0$, $j = 1, 2, ..., m-1$ y $g^{(m)}(x^*) \neq 0$, entonces la iteración de punto fijo siempre converge en algún vecindario de x^* ([24], [20]).

Ejemplo 4.10

Aunque $|g'(x^*)| \not< 1$, podría haber convergencia: Basta con que algún $x_i = x^*$: Sea $f(x) = \begin{cases} 2x & \text{si} \quad |x| \leq 1 \\ 0 & \text{si} \quad |x| > 1 \end{cases}$

f' es continua en cualquier vecindario de $x = 0$ y $f'(0) = 2$ pero la iteración de punto fijo converge para cualquier $x_0 \in [-1,1]$ pues para k suficientemente grande, $|2^k x_0| > 1$, así $x_k = x_{k+1} = ... = 0$.

Ejemplo 4.11 (*)

La sucesión $a_1 = \sqrt{2}$, $a_2 = \sqrt{2}^{\sqrt{2}}$, $a_3 = \sqrt{2}^{\sqrt{2}^{\sqrt{2}}}$, ..., converge a 2.

Esta sucesión es convergente y está acotada por 2 : Para probarlo procedemos por inducción. Observe que $a_n = \sqrt{2}^{a_{n-1}}$, $n = 2,3,...$ entonces como la exponencial a^x $(a > 1)$ es creciente, se tiene a_n es creciente: En efecto, $a_2 > a_1$ pues $\sqrt{2}^{\sqrt{2}} > \sqrt{2}$. Además si $a_n > a_{n-1}$ entonces $a_{n+1} = \sqrt{2}^{a_n} > \sqrt{2}^{a_{n-1}} = a_n$

$a_n < 2$, $n = 1,2,...$: En efecto, $a_1 = \sqrt{2} < 2$. Además si $a_n < 2$ entonces $a_{n+1} = \sqrt{2}^{a_n} < \sqrt{2}^2 = 2$.

Para establecer la convergencia, usamos un argumento basado en los teoremas de punto fijo. Si a_n converge a x^* entonces tenemos los dos límites

$$\begin{aligned} \lim_{n\to\infty} (\sqrt{2})^{a_n} &= \sqrt{2}^{x^*} \\ \lim_{n\to\infty} (\sqrt{2})^{a_n} = \lim_{n\to\infty} a_{n+1} &= x^* \end{aligned}$$

con lo que x^* es una de las soluciones de la ecuación $\sqrt{2}^x = x$ en $[0,2]$. Claramente, $x^* = 2$ es una solución. Además es la única solución $[0,2]$ pues la función $g(x) = \sqrt{2}^x$ cumple las condiciones del teorema de punto fijo en este intervalo. En efecto, como g es creciente, $0 < g(x) < g(2) = 2$ si $x \in [0,2]$ y en este mismo intervalo, $|g'(x)| = \sqrt{2}^x \cdot \ln(\sqrt{2}) \leq 2\ln(\sqrt{2})$ con lo que $|g'(x)| < 1$ en $]0,2[$ por ser g' creciente.

Entonces la solución $x^* = 2$ es la única posible.

Teorema 4.6 (Orden de Convergencia).

Sea x^* un punto fijo de g y sea $I_\varepsilon = \{x \in \mathbb{R} : |x - x^*| \leq \varepsilon\}$. Asumamos que $g \in C^p[I_\varepsilon]$ y además

$$g'(x^*) = g''(x^*) = ... = g^{(p-1)}(x^*) = 0, \quad \text{pero} \quad g^{(p)}(x^*) \neq 0$$

Entonces, si la iteración de punto fijo converge a x^*, el orden de convergencia es p.

Prueba

Hay que mostrar que

$$\lim_{n\to\infty} \frac{|x_{n+1} - x^*|}{|(x_n - x^*)^p|} = K,\ 0 < K < \infty$$

Por el teorema de Taylor

$$\begin{aligned} g(x_n) &= g(x^*) + (x_n - x^*)g'(x^*) + ... + \frac{(x_n - x^*)^{p-1}}{(p-1)!} g^{(p-1)}(x^*) + \frac{(x_n - x^*)^p}{p!} g^{(p)}(\xi_n) \\ &= g(x^*) + \frac{(x_n - x^*)^p}{p!} g^{(p)}(\xi_n) \end{aligned}$$

con ξ_n entre x^* y x_n. Puesto que $g(x_n) = x_{n+1}$ y $g(x^*) = x^*$ entonces

$$\frac{x_{n+1} - x^*}{(x_n - x^*)^p} = \frac{1}{p!} g^{(p)}(\xi_n)$$

luego, conforme $x_n \to x^*$, $\xi_n \to x^*$ pues ξ está entre x_n y x^*. Por la continuidad de $g^{(p)}$ en x^* obtenemos

$$\lim_{n\to\infty} \frac{x_{n+1} - x^*}{(x_n - x^*)^p} = \frac{1}{p!} g^{(p)}(x^*) \neq 0$$

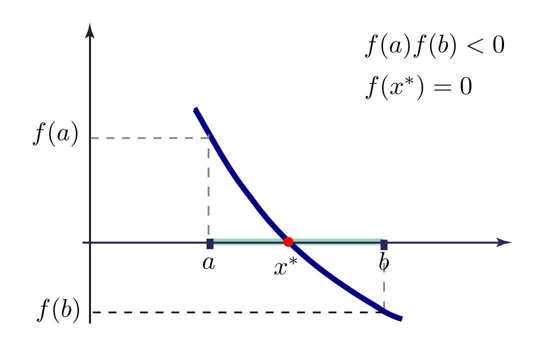
Ejemplo 4.12

El teorema (4.6) lo podemos usar si conocemos x^*.

Ya sabíamos que si $x_0 > 0$ y $A > 0$, $x_{n+1} = 0.5(x_n + A/x_n)$ converge a $\sqrt{A}$. Ahora podemos establecer el orden de convergencia de una manera más sencilla: Sea $g(x) = 0.5(x + A/x)$. g y g' cumplen las condiciones del teorema en un vecindario $I_\varepsilon = \{x \in \mathbb{R} : |x - \sqrt{A}| \leq \varepsilon\}$: Basta tomar $\varepsilon > 0$ tal que $\sqrt{A} - \varepsilon > 0$. Ahora, como $g'(\sqrt{A}) = 0$ pero $g''(\sqrt{A}) \neq 0$, el orden de convergencia de esta sucesión es $q = 2$.

4.5 El método de Bisección

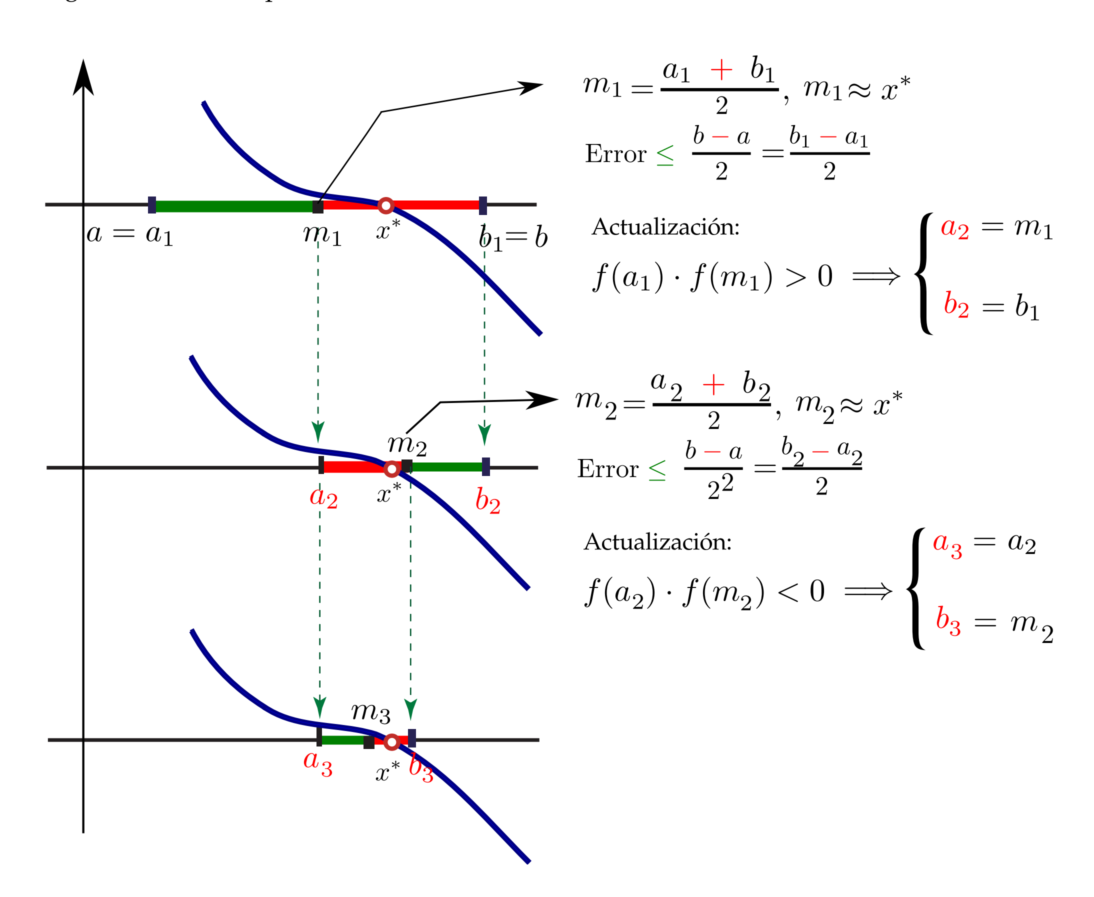
Este es uno de los métodos más sencillos y de fácil intuición, para resolver ecuaciones en una variable. Se basa en el Teorema de los Valores Intermedios, el cual establece que toda función continua f en un intervalo cerrado $[a,b]$ ($f \in C[a,b]$) toma todos los valores que se hallan entre $f(a)$ y $f(b)$. Esto es, que todo valor entre $f(a)$ y $f(b)$ es la imagen de al menos un valor en el intervalo $[a,b]$.



En caso de que $f(a)$ y $f(b)$ tengan signos opuestos (es decir, $f(a) \cdot f(b) < 0$), el valor cero sería un valor intermedio entre $f(a)$ y $f(b)$, por lo que con certeza existe un x^* en $[a,b]$ que cumple $f(x^*) = 0$. De esta forma, se asegura la existencia de al menos una solución de la ecuación $f(x) = 0$.

El método consiste en lo siguiente: Supongamos que en el intervalo $[a,b]$ hay un cero de f. Calculamos el punto medio $m = (a+b)/2$ del intervalo $[a,b]$. A continuación calculamos $f(m)$. En caso de que $f(m)$ sea igual a cero, ya hemos encontrado la solución buscada. En caso de que no lo sea, verificamos si $f(m)$ tiene signo opuesto al de $f(a)$. Se redefine el intervalo $[a,b]$ como $[a,m]$ o $[m,b]$ según se haya determinado en cuál de estos intervalos ocurre un cambio de signo. A este nuevo intervalo se le aplica el mismo procedimiento y así, sucesivamente, iremos encerrando la solución en un intervalo cada vez más pequeño, hasta alcanzar la precisión deseada.

En la siguiente figura se ilustra el procedimiento descrito.



El procedimiento construye tres sucesiones a_n, b_n y m_n,

- Para $k = 1,2,...,$ $\quad m_k = \dfrac{b_k + a_k}{2}$ y $[a_k, b_k] = \begin{cases} [m_{k-1}, b_{k-1}] & \text{si} \quad f(a_{k-1})f(m_{k-1}) > 0 \\ [a_{k-1}, m_{k-1}] & \text{si} \quad f(a_{k-1})f(m_{k-1}) < 0 \end{cases}$

- Estimación del error: El error exacto en el $k-$ésimo paso es $|m_k - x^*|$. Geométricamente se puede ver que esto es menos que la mitad del intervalo $[a_k, b_k]$, es decir

$$|m_k - x^*| \leq \frac{b_k - a_k}{2} = \frac{b-a}{2^k}$$

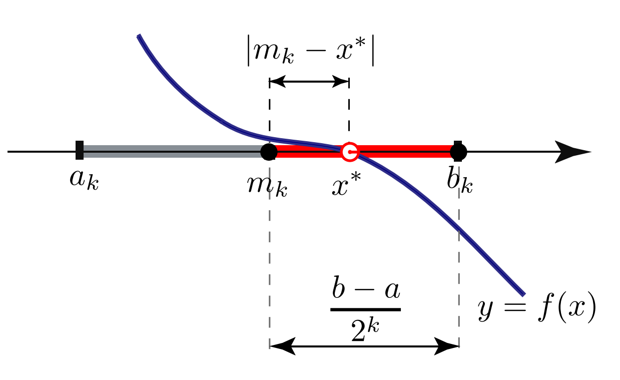


Figura 4.6 Estimación del error en bisección.

Ejemplo 4.13

Aplicar el método de bisección a la ecuación $x^2 = \cos(x) + 1$ en $[1,2]$

- Debemos reescribir la ecuación como $x^2 - \cos(x) - 1 = 0$. En este caso, $f(x) = x^2 - \cos(x) - 1$. Esta función tiene un cero en el intervalo $[1,2]$ pues, efectivamente $f(1) \cdot f(2) = -1.84575... < 0$.

  Calculemos ahora a_k, b_k y m_k así como la estimación del error.

$k = 1:$ $\quad a_1 = 1, \ b_1 = 2$ y $m_1 = \dfrac{a_1 + b_1}{2} = 1.5.$ $\quad$ Error ≤ 0.5

$k = 2:$ $\quad f(a_1) \cdot f(m_1) = -0.637158 < 0,$

$a_2 = 1, \ b_2 = 1.5$ y $m_2 = \dfrac{a_2 + b_2}{2} = 1.25.$ $\quad$ Error ≤ 0.25

$k = 3:$ $\quad f(a_2) \cdot f(m_2) = -0.13355064 < 0,$

$a_3 = 1, \ b_3 = 1.25$ y $m_3 = \dfrac{a_3 + b_3}{2} = 1.125.$ $\quad$ Error ≤ 0.125

$k = 4:$ $\quad f(a_3) \cdot f(m_3) = 0{,}02740730 > 0,$

$a_4 = 1.125, \ b_4 = 1.25$ y $m_4 = \dfrac{a_4 + b_4}{2} = 1{,}1875.$ $\quad$ Error ≤ 0.0625

$\vdots$

$k = 44:$ $\quad m_{44} = 1.17650193990184.$ $\quad$ Error $\leq 2.84 \times 10^{-14}$

- Así, $m_{44} = 1.17650193990184$ aproxima el cero de $f(x) = x^2 - \cos(x) - 1$ en $[1,2]$ con un error $\leq 2{,}84 \times 10^{-14}$

Ejemplo 4.13 (continuación).

Podemos poner estos cálculos (junto con otros adicionales) en una tabla

k	a_k	b_k	m_k	Error Estimado
1	1	2	1.5	0.5
2	1	1.5	1.25	0.25
3	1	1.25	1.125	0.125
4	1.125	1.25	1.1875	0.0625
...	...			
44	1.176501939	1.176501939	1.176501939	2.84217×10^{-14}

4.6 Algoritmo e Implementación.

En la implementación del método de bisección, en vez de usar $f(a)f(m) < 0$ ponemos $\text{sgn}(f(a)) <> \text{sgn}(f(m))$, de esta manera nos ganamos una multiplicación (que es claramente innecesaria).

En bisección es mejor calcular m como $m = a + (b-a)/2$. Con esto somos consistentes con la estrategia general (en análisis numérico) de calcular una cantidad agregando una corrección a la aproximación anterior. Además ganamos algo en precisión.

El criterio de parada es: Detenerse en m_k si $\dfrac{b_k - a_k}{2} = \dfrac{b-a}{2^k} \leq \delta$

Algoritmo 4.2: Algoritmo de Bisección.

Datos: a, b, δ y f continua en [a,b] con $f(a)f(b) < 0$.
Salida: Una aproximación m de un cero x^* de f en $]a,b[$.

$k = 0$;
repeat
  $m = a + 0.5(b-a)$;
  $dx = (b-a)/2$;
  if $\text{Sgn}(f(a)) <> \text{Sgn}(f(m))$ **then**
    $b = m$;
  else
    $a = m$
  $k = k+1$
until $dx \leq \delta$;
return m

Implementación Basic. La implementación Basic (Excel, OpenOffice, Libreoffice) requiere definir la función f en el código. La implementación es directa.

Código VBA 4.4: Método de Bisección

```
Function f(x)
 f=0.2*Sin(16*x)-x+1.75 'Tiene un único cero en [1,2]
End Function
'-----------------------------------------------------------

Function Biseccion(a,b,delta)
Dim k, a1,b1, m, dx

k = 0
a1 = a
b1 = b
Do
    m = a1 + 0.5 * (b1 - a1)
       dx = (b1 - a1) / 2
    If Sgn(f(a1)) <> Sgn(f(m)) Then
          b1 = m
    Else: a1 = m
    End If
 k = k + 1
 Loop Until dx <= delta
 Biseccion = m
End function
```

Hoja para Bisección. A continuación aparece la "página principal" de una hoja para Excel y otra para Libreoffice (y Openffice). La implementación recibe los valores a, b y δ.

	a	b	Tolerancia	ak	mk	bk	Error Estimado
4					Bisección		
5							
6							
7	a	b	Tolerancia	ak	mk	bk	Error Estimado
8	-0,69	-0,67	0,0000005	-0,6900000000000000	-0,6800000000000000	-0,6700000000000000	0,0100000000000000
9				-0,6900000000000000	-0,6850000000000000	-0,6800000000000000	0,0050000000000000
10				-0,6850000000000000	-0,6825000000000000	-0,6800000000000000	0,0025000000000000
11				-0,6825000000000000	-0,6812500000000000	-0,6800000000000000	0,0012500000000000
12				-0,6825000000000000	-0,6818750000000000	-0,6812500000000000	0,0006250000000000
13				-0,6825000000000000	-0,6821875000000000	-0,6818750000000000	0,0003125000000000
14				-0,6825000000000000	-0,6823437500000000	-0,6821875000000000	0,0001562500000000
15				-0,6823437500000000	-0,6822656250000000	-0,6821875000000000	0,0000781250000000
16				-0,6823437500000000	-0,6823046875000000	-0,6822656250000000	0,0000390625000000

Figura 4.7 Hoja Excel para Bisección.



Software: Cuadernos LibreOffice y Excel

Código VBA 4.5: Cuaderno Excel

```
Private Sub CommandButton1_Click() 'botón del cuaderno
Dim a, b, delta
    a     = Cells(8, 1)
    b     = Cells(8, 2)
    delta = Cells(8, 3)
```

```
   Call Biseccion(a, b, delta)
End Sub
```

Código VBA 4.6: Cuaderno Libreoffice

```
Sub Main  'Subrutina principal
         'Cargar la biblioteca BblMatematica
   BasicLibraries.loadLibrary("BblMatematica" )
   Dim a , b, delta

         'function cells está en BblMatematica
  a     =  cells("A8").value
  b     =  cells("B8").Value
  delta =  cells("C8").Value
  CleanRange(3,7,4) 'Limpiar corrida anterior. Esta función está en BblMatematica
  Call Biseccion(a,b, delta)
End Sub
```

4.7 Bisección: Criterio de Parada y Número de Iteraciones.

Sea x^* es el único cero de f en $[a,b]$. En la k-ésima iteración, al aproximar x^* con m_k se tiene que

$$|m_k - x^*| \leq \frac{b-a}{2^k}, \quad k = 1,2,...$$

entonces,

- Si tenemos una *tolerancia* $\delta > 0$, y si queremos cortar la sucesión m_n en la k-ésima iteración m_k de tal manera que $|m_k - x^*| \leq \delta$ entonces podemos estimar el número k de iteraciones con

$$|m_k - x^*| \leq \frac{b-a}{2^k} \leq \delta$$

Tomando logaritmo natural a ambos lados de $\dfrac{b-a}{2^k} \leq \delta$ obtenemos

$$k \geq \ln\left(\frac{b-a}{\delta}\right) / \ln 2 \quad \text{iteraciones}$$

excepto que en algún momento $f(m_j) = 0$ para algún $j < k$

De aquí podemos deducir que si $|b-a| < 1$ y si $\delta = 2^{-52}$ entonces el número de iteraciones necesarias para alcanzar esta tolerancia δ es como mínimo $k = 52$.

- Observe que $2^{-3.4} \approx 10^{-1}$ y en general $2^{-3.4 \cdot d} \leq 10^{-d}$. Como $|m_k - x^*| \leq \dfrac{b-a}{2^k}$, esto nos dice que si $b-a \leq 1$ entonces bisección se gana un dígito decimal cada 3.4 iteraciones aproximadamente.

Ejemplo 4.14

Al aplicar el algoritmo de bisección a una función continua f con un único cero en el intervalo $[-2,1]$, si queremos que el error de aproximación sea $\leq 0.00005 = 0.5 \times 10^{-4}$, el número k de iteraciones debe cumplir

$$k \geq \ln\left(\frac{3 \cdot 10^4}{0.5}\right) / \ln 2 = 15.873$$

por lo que deben realizarse por lo menos 16 iteraciones. Verifíquelo!

Notas.

a.) El método de bisección, la única información que usa es el signo de f. El número de iteraciones no depende, en general, de la función f, depende del intervalo y la tolerancia δ. La única manera de que el número de iteraciones sea menor es que se dé el caso $f(m_j) = 0$, para algún $j < k$.

b.) Aunque el método de bisección es *robusto* al nivel de la precisión de la máquina, puede pasar que la aproximación *no* quede numéricamente bien determinada si la función es muy "aplanada".

EJERCICIOS

4.12 Si f tienen un único cero en $[-2,5]$, ¿Cuántas iteraciones de bisección se deben hacer para aproximar este cero con error absoluto $\leq 0.5 \times 10^{-4}$?

4.13 Resuelva $e^{3(x-1)} - \ln(x-1)^2 + 1 = 0$ con al menos cinco decimales exactos.

4.14 Resuelva $e^{3x} - \ln(x^2 + 1) - 30 = 0$ con al menos cinco decimales exactos.

4.15 Como $\sqrt{A}$ es una solución de la ecuación $x^2 - A = 0$, podemos usar el método de bisección para estimar $\sqrt{A}$. Estimar $\sqrt{3}$ y $\sqrt{1000999}$ con al menos cinco decimales exactos.

4.16 $(x-1)^2 = 0$ tiene claramente una raíz en $[0,2]$. ¿Podemos usar bisección para aproximarla?

4.17 Sea $f(x) = x^3 - 2\cos(x) - 3$. Según el teorema de Taylor

$$f(\alpha) = f(x) + f'(x)(\alpha - x) + \frac{1}{2}(\alpha - x)^2 f''(\xi), \quad \text{con } \xi \text{ entre } x \text{ y } \alpha$$

Si $\alpha = \pi/4$ y $x = 1$, aproximar ξ con un error $\leq 0.5 \times 10^{-8}$.

4.18 $x = 2$ es un cero del polinomio $P(x) = -1536 + 6272\,x - 11328\,x^2 + 11872\,x^3 - 7952\,x^4 + 3528\,x^5 - 1036\,x^6 + 194\,x^7 - 21\,x^8 + x^9$. Aproxime esta raíz, con un intervalo adecuado.

4.19 $x = 1$ es un cero del polinomio $P(x) = 2 - 19\,x + 81\,x^2 - 204\,x^3 + 336\,x^4 - 378\,x^5 + 294\,x^6 - 156\,x^7 + 54\,x^8 - 11\,x^9 + x^{10}$. Aproxime esta raíz, con un intervalo adecuado.

4.20 Verifique que $|m_k - m_{k-1}| = \dfrac{b_k - a_k}{2}$ si $k \geq 2$.

4.21 (*) Supongamos que modificamos el método de bisección de la siguiente manera: En cada iteración, en vez de dividir el intervalo en dos, lo dividimos en tres partes iguales y, como en bisección, nos quedamos con el intervalo donde hay cambio de signo para proseguir con la nueva división en tres partes iguales y así sucesivamente. ¿Mejora este procedimiento la eficiencia de bisección?

4.8 Bisección: Teorema de Convergencia. Orden de Convergencia.

Teorema 4.7 (Convergencia del método de bisección).

Sea $f \in C[a,b]$, con $f(a)f(b) < 0$. Sea $\{m_n\}_{n=1}^{\infty}$ la sucesión de puntos medios de los subintervalos generados por el método de bisección descrito anteriormente. Entonces existe un número $x^* \in [a,b]$ tal que $f(x^*) = 0$ y además

$$|x^* - m_{k+1}| \leq \frac{b-a}{2^{k+1}}, \quad k = 0, 1, ...$$

en particular, $\lim\limits_{k\to\infty} m_k = x^*$.

Prueba. En la $k-$ésima iteración, el método de bisección ha construido una sucesión de intervalos $[a_k, b_k] \subseteq [a_{k-1}, b_{k-1}] \subseteq ... \subseteq [a_0, b_0]$ con las siguientes propiedades,

$$a_0 \leq a_1 \leq ... \leq a_k \leq b_0 \tag{4.5}$$

$$a_0 \leq b_k \leq ... \leq b_1 \leq b_0 \tag{4.6}$$

$$b_{k+1} - a_{k+1} = \frac{1}{2}(a_k - b_k), \;\; k \geq 0 \tag{4.7}$$

Las ecuaciones (4.5) y (4.6) indican que las sucesiones $\{a_n\}$ y $\{b_n\}$ son monótonas y acotadas, por tanto convergen. Aplicando repetidamente (4.7) obtenemos

$$b_{k+1} - a_{k+1} = \frac{1}{2^{k+1}}(a_0 - b_0) \tag{4.8}$$

entonces,

$$\lim_{k\to\infty}(b_{k+1} - a_{k+1}) = 0 \implies \lim_{k\to\infty} b_k = \lim_{k\to\infty} a_k$$

Si $x^* = \lim_{k\to\infty} a_k$, entonces $b_k \leq m_{k+1} \leq a_k \implies \lim_{k\to\infty} m_k = x^*$.

En $[a,b]$ podrían haber varios ceros!. Nos interesa un cero en particular: x^* es un cero de f pues aplicando límites a ambos lados de $f(a_n)f(b_n) \leq 0$ obtenemos $[f(x^*)]^2 \leq 0$ de donde $f(x^*) = 0$.

Finalmente, para todo $k \geq 0$, $x^* \in [a_k, b_k] = [a_k, m_{k+1}] \cup [m_{k+1}, b_k]$. Como x^* debe estar en alguna de las dos mitades entonces, usando (4.8) obtenemos,

$$|x^* - m_{k+1}| \leq \frac{1}{2}|b_k - a_k| \leq \frac{1}{2}\frac{1}{2^k}(a_0 - b_0) = \frac{b-a}{2^{k+1}}$$

Orden de Convergencia. El método de bisección es un método seguro pero relativamente lento. En general, su orden de convergencia *no es lineal* ([10]) aunque, *en promedio*, se comporta como si lo fuera pues, si $b - a \leq 1$, gana un dígito decimal cada 3.4 iteraciones aproximadamente.

4.9 Acerca del Criterio de Parada en métodos iterativos.

No todos los métodos que estamos estudiando tienen una manera sencilla de estimar el error en cada aproximación. La práctica computacional requiere de un criterio de parada que termine la iteración. Esto fue sencillo de establecer para bisección. Pero en los métodos que siguen no es así.

El criterio de parada que vamos a usar es: *parar* la iteración una vez que se alcance la exactitud deseada.

Idealmente la iteración se detiene tan pronto como $|x_n - x^*|$ sea menor que una tolerancia pre-establecida. Como x^* no es conocida, usualmente se reemplaza $|x_n - x^*|$ por $|x_n - x_{n-1}|$. Una opción prudente es establecer como criterio de parada

$$|x_n - x_{n-1}| \leq \delta_1 |x_n| + \delta_2$$

Si $\delta_1 = 0$ obtenemos una estimación del error absoluto y si $\delta_2 = 0$ obtenemos una estimación del error relativo. Así que podemos poner $\delta_1 = \delta_2 = \delta$. Con esta elección si $|x_n|$ es pequeño o moderadamente grande, efectivamente controlamos el error absoluto mientras que si $|x_n|$ es muy grande el que se ve controlado es el error relativo.

A continuación se hacen algunas observaciones sobre algunas variantes que se usan como criterio de parada.

a.) $|x_k - x_{k-1}| < \delta_1$. El error absoluto ($|x_k - x^*|$) en la k-ésima aproximación se estima con $|x_k - x_{k-1}|$, es decir se establece una tolerancia δ_1 como criterio de parada: $|x_k - x_{k-1}| < \delta_1$.

Un peligro con este criterio de parada es que $|x_k - x_{k-1}|$ puede ser pequeño pero $f(x_k)$ no tan pequeño como uno desearía o aún que x_k todavía no esté cercano a la raíz (ver el ejemplo (4.28)).

b.) $|f(x_k)| < \epsilon$. Un peligro con este criterio de parada es que aunque $f(x_k)$ sea pequeño, puede ser que x_k no esté tan cerca de la raíz como uno desearía.

c.) $\dfrac{2|x_k - x_{k-1}|}{|x_k| + \varepsilon} < \delta_2$. El error relativo en la k-ésima aproximación se estima con $\dfrac{2|x_k - x_{k-1}|}{|x_k| + |x_{k-1}|}$ o algo como $\dfrac{2|x_k - x_{k-1}|}{|x_k| + \varepsilon}$, donde ε es un número real suficientemente pequeño. El error relativo nos da una mejor visión del comportamiento de las diferencias $|x_k - x_{k+1}|$ si x_k está tomando valores muy pequeños o valores muy grandes.

d.) También se usa el siguiente criterio: detener el proceso *la primera vez* que

$$|x_{k-2} - x_{k-1}| > |x_{k-1} - x_k| \;\text{ y }\; |x_{k-1} - x_k| < \delta$$

Este criterio es útil sobre todo cuando $f'(x_j)$ es tan pequeño que el error se ve severamente afectado por el redondeo de la máquina y éstas diferencias varían aleatoriamente. Este tipo de comportamiento se da por ejemplo, cuando aproximamos la raíz $x = 2$ de $P(x) = -1536 + 6272\,x - 11328\,x^2 + 11872\,x^3 - 7952\,x^4 + 3528\,x^5 - 1036\,x^6 + 194\,x^7 - 21\,x^8 + x^9$ (figura 4.8).

En estos casos, no se puede hacer algo mejor con ningún método de este capítulo ni aún con métodos especializados en polinomios, como es el caso de P. Se debe aceptar el último valor x_n calculado como satisfactorio. Una alternativa que evita estos problemas (basada en ideas más avanzadas) se puede ver en [17]

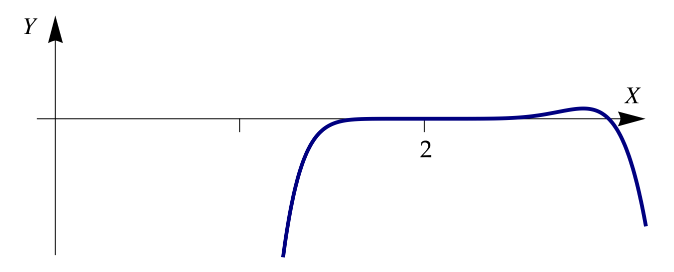


Figura 4.8 Gráfica de $P(x)$.

En general, podríamos usar un criterio de parada que tome en cuenta solo uno de estos criterios o que tome en cuenta estos tres criterios, es decir, aceptamos como aproximación de la raíz a x_k si

$$|x_k - x_{k-1}| \leq \delta\,(|x_k| + 1) \text{ y } |f(x_k)| < \epsilon$$

junto con un número máximo de iteraciones.

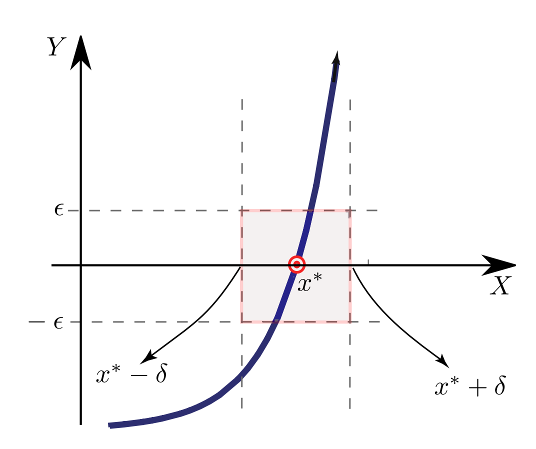


Figura 4.9 Región para un criterio de parada que toma en cuenta los tres criterios anteriores

4.10 El Método de Newton

El método de Newton (llamado a veces método de Newton-Raphson[7]) es uno de los métodos que muestra mejor velocidad de convergencia llegando (bajo ciertas condiciones) a duplicar, en cada iteración, los decimales exactos.

Si f es una función tal que f, f' y f'' existen y son continuas en un intervalo I y si un cero x^* de f está en I, se puede construir una sucesión $\{x_n\}$ de aproximaciones, que converge a x^* (bajo ciertas condiciones) de la manera que se describe a continuación: Si x_0 está *suficientemente cercano* al cero x^*, entonces supogamos que h es la *corrección* que necesita x_0 para alcanzar a x^*, es decir, $x_0 + h = x^*$ y $f(x_0 + h) = 0$. Como

$$0 \;=\; f(x_0 + h) \approx f(x_0) + hf'(x_0)$$

entonces 'despejamos' *la corrección* h,

$$h \approx -\frac{f(x_0)}{f'(x_0)}$$

De esta manera, una aproximación *corregida* de x_0 sería

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

Aplicando el mismo razonamiento a x_1 obtendríamos la aproximación $x_2 = x_1 - \dfrac{f(x_1)}{f'(x_1)}$ y así sucesivamente.

Geométricamente se vería así: Partiendo de una aproximación x_0 de un cero x^* de f, entonces x_1 es la intersección, de la recta tangente a f en x_0, con el eje X. Cuando se ha calculado una aproximación x_n, la siguiente aproximación x_{n+1} se obtiene hallando la intersección con el eje X de la recta tangente en el punto $(x_n, f(x_n))$. El proceso se muestra en las figuras (4.10), (4.11) y (4.12).

La intersección de la tangente con el eje X se obtiene resolviendo $y = 0$. Como la ecuación de una recta de pendiente m que pasa por (x_0, y_0) es $y = m(x - x_0) + y_0$ entonces la ecuación de la recta tangente en (x_n, y_n) es

$$y = f'(x_n)(x - x_n) + y_n$$

Al despejar el valor de x, se obtiene x_{n+1} :

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

En general, en la fórmula $x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)}$, $f(x_n)$ no presenta grande variaciones mientras que $f'(x_n)$ puede variar fuertemente: Si $f'(x_n)$ es grande entonces la *corrección* $\dfrac{f(x_n)}{f'(x_n)}$ que se le aplica a x_n para aproximar el cero es

[7]



En la literatura inglesa se acostumbra llamar al método de Newton, método de Newton-Raphson. El método de Newton aparece en su libro 'Method of Fluxions" escrito en 1671 pero publicado hasta 1736. El método fue publicado por primera vez en un libro de J. Raphson en 1690. Se dice que Raphson tuvo acceso al manuscrito de Newton. En todo caso ni Newton ni Raphson mencionan las derivadas. El primero que dio una descripción del método de Newton usando derivadas fue T. Simpson en 1740.

pequeña. Por otro lado, si $f'(x_n)$ es pequeña entonces la corrección sería mayor por lo que aproximar un cero de f puede ser un proceso lento o a veces imposible.

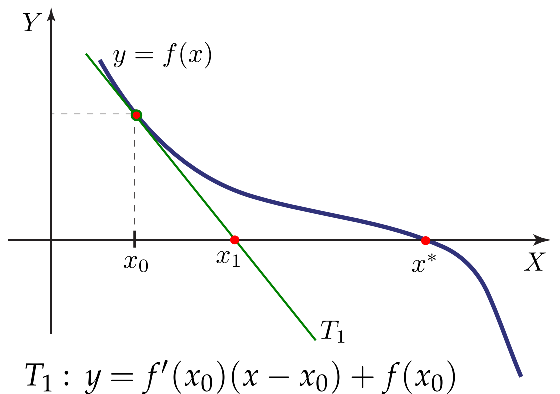


Figura 4.10 Obteniendo x_1

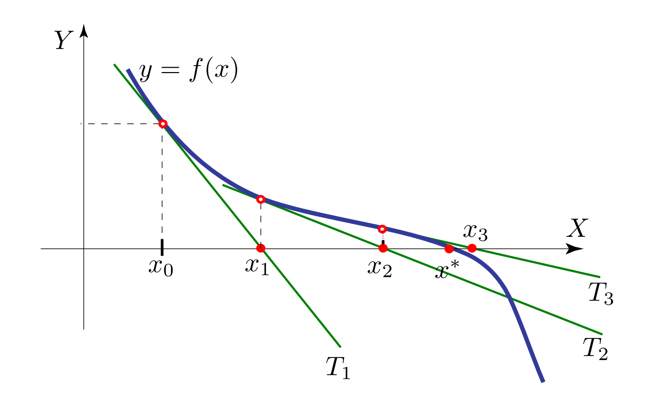


Figura 4.11 Obteniendo $x_1, x_2, \ldots$

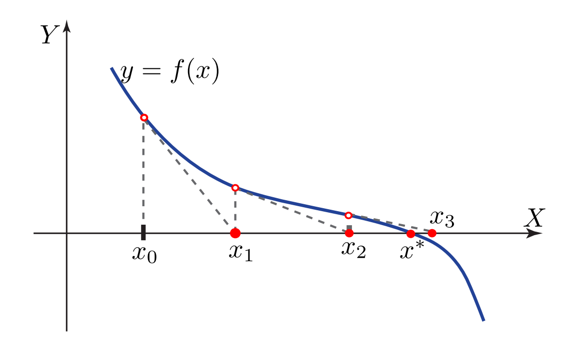


Figura 4.12 Representación simplificada

Ejemplo 4.15

La ecuación $x^2 - \cos(x) - 1 = 0$ tiene una solución x^* en $[1,2]$. Debemos tomar una aproximación inicial, digamos $x_0 = 1.5$.

$f(x) = x^2 - \cos(x) - 1$ y $f'(x) = 2x + \text{sen}(x)$. Entonces *el esquema iterativo* es

$$x_{n+1} = x_n - \frac{x_n^2 - \cos(x_n) - 1}{2x_n + \text{sen}(x_n)}$$

Ejemplo 4.15 (continuación).

$n=0: \; x_0 = 1.5$

$$n=1: \; x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 1.5 - \frac{f(1.5)}{f'(1.5)} = 1.204999.$$

Error $\leq |x_1 - x_0| = 0{,}295$

$$n=2: \; x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 1.204999 - \frac{f(1.204999)}{f'(1.204999)} = 1.176789.$$

Error $\leq |x_2 - x_1| = 0{,}0282$

$$n=3: \; x_3 = x_2 - \frac{f(x_2)}{f'(x_2)} = 1.176789 - \frac{f(1.176789)}{f'(1.176789)} = 1.1765019.$$

Error $\leq |x_3 - x_2| = 0{,}000287$

Podemos tabular esta información en una tabla agregando más iteraciones. Si usamos Excel obtenemos

x_{n+1}	$\lvert x_{n+1} - x_n \rvert$
1.20499955540054	0.295000445
1.17678931926590	0.028210236
1.17650196994274	0.000287349
1.17650193990183	3.004×10^{-8}
1.17650193990183	4.440×10^{-16}

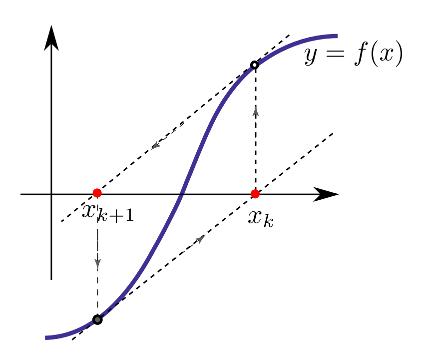
Compare con la solución $x^* = 1.17650193990183$ (exacta en sus catorce decimales) y observe como, aproximadamente, se *duplican* los decimales correctos desde la segunda iteración.

Las limitaciones numéricas de Excel hacen que la iteración 4 y 5 repitan valor. Usando *Mathematica* podemos obtener la aproximación (con 34 dígitos) usando el comando `FindRoot` (este comando utiliza un método hibrido: el método 'amortiguado' de Newton, el método de la secante y el método de Brent.)

```
In[]:=  FindRoot[x^2 - Cos[x] - 1 == 0, {x, 1.5},
                 AccuracyGoal -> 20,
                 WorkingPrecision -> 34, MaxIterations -> 6]

Out[]=  x-> 1.176501939901832400447377268731041
```

Algunas veces el método de Newton no converge. En las gráficas de la figura que sigue se muestran dos situaciones donde no hay convergencia.



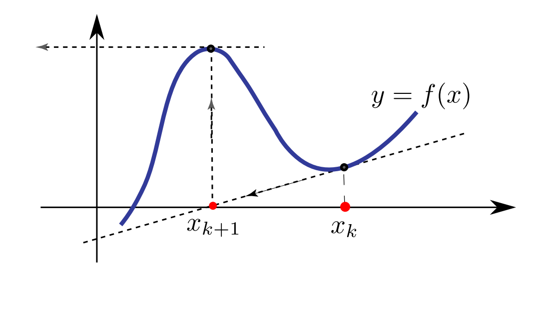
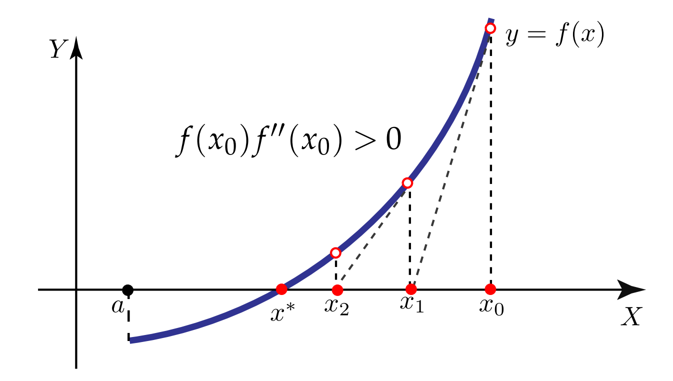


Figura 4.13 Situaciones donde el método de Newton no converge

Convergencia global. La convergencia del método de Newton *es local*, es decir, como aproximación inicial de debe elegir un x_0 que este "suficientemente cercano" a x^*. Sin embargo hay teoremas que dan criterios para la convergencia global (para ciertos tipos de funciones).Por ejemplo, si en el intervalo $I = [a, b]$ f tiene un único cero y si f' y f'' conservan el signo, entonces una buena aproximación inicial es cualquier $x_0 \in [a, b]$ para el cual $f(x_0)f''(x_0) > 0$ (ver figura).



En el ejemplo (4.15), $x_0 = 1.5$ fue una buena aproximación pues x_0 está en $[1,2]$ y en este intervalo f' y f'' son positivas y además $f(1.5)f''(1.5) > 0$.

Ejemplo 4.16

Uno de los peligros del método de Newton es *no tomar* una aproximación inicial x_0 suficientemente cercana a x^*. Para ver esto consideremos la ecuación $x^{20} - 1 = 0$, $x > 0$. Esta ecuación tiene una única solución $x^* = 1$. El esquema iterativo es

$$x_{n+1} = \frac{19}{20}x_n + \frac{1}{20x_n^{19}}$$

Ejemplo 4.16 (continuación).

Si tomamos la aproximación inicial $x_0 = 0.5$ entonces

x_n		
x_1	=	26214.9
x_2	=	24904.1
x_3	=	23658.9
x_4	=	22476.
x_5	=	21352.2
...		
x_{200}	=	1.01028

De hecho, si n es grande entonces $x_{n+1} \approx \dfrac{19}{20}x_n$, es decir, la corrección es algo pequeña (casi deja igual a x_{n+1}) y toma unas 200 iteraciones llegar cerca de la raíz $x^* = 1$.

Si tomamos la aproximación inicial $x_0 = 0.92$ entonces las cosas cambian dramaticamente

x_n		
x_1	=	1.11778
x_2	=	1.06792
x_3	=	1.02887
x_4	=	1.00654
x_5	=	1.00039
x_6	=	1.0000014308157694
x_7	=	1.0000000000194484

Ejemplo 4.18 (Sucesión divergente).

Si a la ecuación $xe^{-x} = 0$ le aplicamos el método de Newton con $x_0 = 2$, obtenemos una sucesión divergente. La sucesión se aleja rápidamente de la raíz $x^* = 0$. En cambio, si ponemos $x_0 = -0.4$, obtenemos una rápida convergencia.

$x_0 = 2$ x_n	Error estimado	$x_0 = -0.4$ x_n	Error estimado
4	2	-0.114285714	0.285714286
5.333333333	1.333333333	-0.011721612	0.102564103
6.564102564	1.230769231	-0.000135804	0.011585807
7.743826066	1.179723502	-1.84403$\times 10^{-8}$	0.000135786
8.892109843	1.148283777	-3.40045$\times 10^{-16}$	1.84403$\times 10^{-8}$
10.01881867	1.126708829	-1.47911$\times 10^{-31}$	3.40045$\times 10^{-16}$

Ejemplo 4.17 (Ciclos)

Consideremos la ecuación $\text{sen}(x) = 0, \;\; |x| < \frac{\pi}{2}$. Esta ecuación solo tiene la solución $x^* = 0$. El esquema iterativo es

$$x_{n+1} = x_n - \tan(x_n)$$

Si tomamos x_0 tal que $\tan(x_0) = 2x_0$ entonces

$$x_1 = x_0 - \tan(x_0) = -x_0$$

$$x_2 = -x_0 - \tan(-x_0) = x_0$$

es decir, entramos en un ciclo (figura 4.14). Observe que

$$\tan(x_0) = 2x_0 \Longrightarrow x_0 = 1.16556...$$

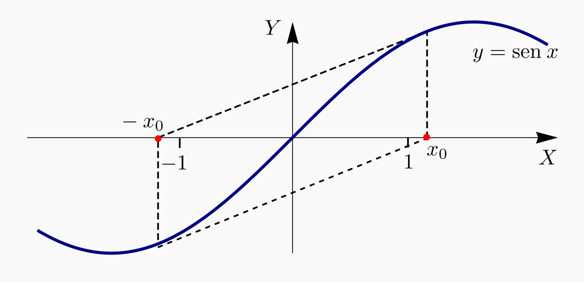


Figura 4.14 Ciclo

4.11 Método de Newton: Algoritmo e Implementación.

Algoritmo 4.3: Método de Newton

Datos: $f \in C^2[a,b], \; x_0, \delta,$ maxItr.
Salida: Si la iteración converge, una aproximación x_n de un cero de f en $[a,b]$ y una estimación del error.

$k = 0$;
$x_k = x_0$;
repeat
  $dx = \dfrac{f(x_k)}{f'(x_k)}$;
  $x_{k+1} = x_k - dx$;
  $x_k = x_{k+1}$;
  $k = k+1$;
until $dx \leq \delta\,(|x_{k+1}|+1)$ *or* $k \leq$ maxItr ;
return x_k y dx

Implementación en Basic. Una implementación directa en Basic (Excel, OpenOffice, LibreOffice) requiere definir la función f y f' por separado. Observe que $|x_{n+1} - x_n| = \left|\dfrac{f(x_n)}{f'(x_n)}\right|$.

Código VBA 4.7: Método de Newton para ecuaciones no lineales.

```
Function f(x)
   f= x^3+x+1 'Tiene un único cero en [-1,0]
End Function
```

```
'-------------------------------------------------
Function Df(x)
     Df= 3*x^2+1
End Function
'-------------------------------------------------
Function MNewton(xx, delta, maxitr)
 Dim x0, x1, k, dx

 x0 = xx
 k=0
 Do
    dx = f(x0)/Df(x0)
    x1 = x0 - dx
    x0 = x1
    k=k+1
 Loop Until abs(dx) < delta Or k >= maxitr

 MNewton = x1

End Function
```

Cuaderno para Excel, LibreOffice u OpenOffice. A continuación se muestra un cuaderno y el código en Excel y en Calc de LibreOffice u OpenOffice.

5	Aproximar x* tal que f(x*) = 0. Método de newton					
6						Método Newton
7		Tolerancia	1,76		Erorr absoluto	
8	x0	delta	MaxIteraciones	xn	Estimado	
9	1,700000000000	0,000000	20,0	1,70000000000000	0,08947509062816	
10				1,78947509062816	0,02736865764232	
11				1,76210643298584	0,00095529086589	
12				1,76306172385173	0,00000042044312	
13				1,76306130340861	0,00000000000007	
14				1,76306130340854	0,00000000000000	

 **Software:** Cuadernos LibreOffice y Excel

Código VBA 4.8: Método de Newton. Cuaderno LibreOffice u OpenOffice

```
Sub Main 'Subrutina principal
        'Cargar la biblioteca BblMatematica
   BasicLibraries.loadLibrary("BblMatematica" )
   Dim x0, delta, maxitr
       'function cells está en BblMatematica
   x0                 =        Cells("A9").Value
   delta         =        Cells("B9").Value
   maxitr        =        Cells("C9").Value
       'Limpieza de celdas del cálculo anterior - Cleanrange está en BblMatematica
   CleanRange(3,9,2)
```

```
   Cells("C7").Value=  MNewton(x0, delta, maxitr)
End Sub
'-----------------------------------------------------------
Function f(x)
...
```

Cuadero para Excel con `clsmathparser`. Esta implementación usa clsMathParser para leer y evaluar la función f y su derivada. La función recibe las funciones f y f' como "string" y los valores x_0, δ y maxItr.

 **Software:** Cuadernos LibreOffice y Excel

Código VBA 4.9: Cuaderno Excel con `clsMathParser`

```
Private Sub CommandButton1_Click()
Dim fx, fpx, x0, delta, maxItr
    fx = Cells(3, 2)     'Leer f(x) en celda (3,2)
    fpx = Cells(4, 2)    'Leer f'(x)
    x0 = Cells(8, 1)
    delta = Cells(8, 2)
    maxItr = Cells(8, 3)
    Call MNewton(fx, fpx, x0, delta, maxItr, 8, 4)
End Sub
'-----------------------------------------------------------
Sub MNewton(fx, fpx, xc, delta, maxItr, fi, co)
Dim f As New clsMathParser
Dim fp As New clsMathParser
Dim k, x0, x1, dx, okfx, okfpx, fx0, fpx0

okfx = f.StoreExpression(fx)
okfpx = fp.StoreExpression(fpx)

If Not okfx Then
    MsgBox ("Error en f: " + f.ErrorDescription)
    Exit Sub
End If

If Not okfpx Then
    MsgBox ("Error en fp: " + fp.ErrorDescription)
    Exit Sub
End If

k = 0
x0 = xc

Do
     fx0 = f.Eval1(x0)
     fpx0 = fp.Eval1(x0)
     x1 = x0 - fx0 / fpx0
```

```
    dx = Abs(x1 - x0)
    x0 = x1
    Cells(fi + k, co) = x1
    Cells(fi + k, co + 1) = dx
k = k + 1
Loop Until dx < delta Or k > maxItr
End Sub
```

EJERCICIOS

4.22 (**Raíz cuadrada**) Como $\sqrt{A}$ es una solución de la ecuación $x^2 - A = 0$, podemos usar el método de Newton para estimar $\sqrt{A}$ (figura 4.15). Como veremos más adelante, la sucesión converge para cualquier $x_0 > 0$

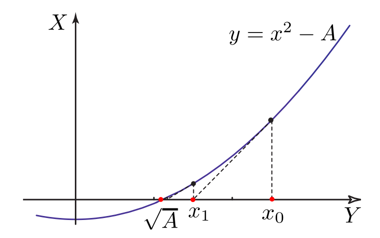


Figura 4.15 La sucesión converge a la raíz para cualquier valor $x_0 > 0$

a) Verifique que la fórmula de iteración para obtener la estimación de la raíz cuadrada es $x_n = 0.5\left(x_{n-1} + \frac{A}{x_{n-1}}\right)$.

b) Estime $\sqrt{2}$ y $\sqrt{1000999}$ con al menos cinco decimales exactos.

4.23 Aproxime $\sqrt[3]{0.00302}$ con al menos cinco cifras significativas

4.24 Aproxime $\sqrt[10]{0.00302}$ con al menos cinco cifras significativas

4.25 Dé una fórmula iterativa para aproximar $\sqrt[N]{a}$ con $a > 0$.

4.26 Resuelva $x^3 = 0$ usando $x_0 = -0.2$. Resuelva la misma ecuación usando bisección con el intervalo $[-0.2, 0.1]$

4.27 Resuelva $f = x^5 - 100 * x^4 + 3995 * x^3 - 79700 * x^2 + 794004 * x - 3160075$ usando $x_0 = 17$. Resuelva usando bisección con $[17, 22.2]$

4.28 (*) En teoría de números es necesario a veces calcular $\lfloor \sqrt{n} \rfloor$ donde n puede ser un número entero grande imposible de manejar aún con doble precisión. En estos casos se implementan bibliotecas para operar con enteros grandes (de precisión infinita, i.e. enteros tan grandes como la máquina soporte). Para calcular $\lfloor \sqrt{n} \rfloor$ se aplica el método de Newton a la ecuación $x^2 - n = 0$ *operando con aritmética entera* en todo momento. El algoritmo que sigue esta basado en el siguiente resultado

Si $n \in \mathbb{N}$, la fórmula recursiva

$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{n}{x_k}\right), \quad k \geq 0,\ x_0 = n.$$

converge (cuadráticamente) a $\sqrt{n}$. Además, si $|x_{k+1} - x_k| < 1$ entonces $\lfloor \sqrt{x_{k+1}} \rfloor = \lfloor \sqrt{n} \rfloor$.

El algoritmo (que funciona con aritmética entera) es el siguiente.

if $n = 1$, salida 1

if $n > 1$

$x_k = n$

$x_{k+1} = \frac{n}{2}$ (división entera)

`while` $x_{k+1} < x_k$

$x_k = x_{k+1}$

$x_{k+1} = \dfrac{x_{k+1} + \dfrac{n}{x_{k+1}}}{2}$ (divisiones enteras)

retorne x_{k+1}

a) Explique porqué es adecuado el criterio de parada.
b) Para decidir el criterio de parada puede utilizar la desigualdad $|x_{k+1} - x_k| < 1$. Utilice esta desigualdad para cambiar el criterio de parada.
c) Implemente el algoritmo.
d) Use la implementación para calcular $\left\lfloor \sqrt{6556426} \right\rfloor$.

4.29 Resuelva $x^3 - 2x - 5 = 0$. Esta ecuación tiene valor histórico: fue la ecuación que usó John Wallis para presentar por primera vez el método de Newton a la academia francesa de ciencias en el siglo XV.

4.30 Resuelva $e^{3(x-1)} - \ln(x-1)^2 + 1 = 0$ con al menos cinco cifras significativas.

4.31 Resuelva $e^{3x} - \ln(x^2 + 1) - 30 = 0$ con al menos cinco cifras significativas.

4.32 Considere la ecuación $x = u\ln(u)$ con $x \geq 0$.

a) Aplique el método de Newton a $f(u)$ y obtenga la expresión para u_{n+1}.

b) Resuelva la ecuación $f(u) = 0$ con $u_0 = 0.5$

4.33 $x = 2$ es un cero del polinomio $P(x) = -1536 + 6272x - 11328x^2 + 11872x^3 - 7952x^4 + 3528x^5 - 1036x^6 + 194x^7 - 21x^8 + x^9$. Aproxime esta raíz con una aproximación inicial adecuada.

4.34 $x = 1$ es un cero del polinomio $P(x) = 2 - 19x + 81x^2 - 204x^3 + 336x^4 - 378x^5 + 294x^6 - 156x^7 + 54x^8 - 11x^9 + x^{10}$. Aproxime esta raíz con una aproximación inicial adecuada.

4.35 Para las siguientes ecuaciones, haga una corrida en Excel para las aproximaciones iniciales que se dan. ¿Es posible encontrar otra aproximación inicial para obtener convergencia eficiente?

a) (Sucesión oscilante y divergente). $\arctan(x) = 0$ con $x_0 = 1.5$

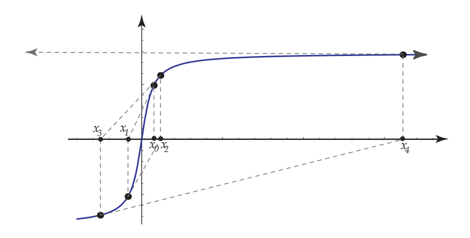


Figura 4.16 $f(x) = \arctan(x)$

b) (Sucesión rápidamente convergente). $x^2 - \cos(x) - 1 = 0.$ con $x_0 = 6$

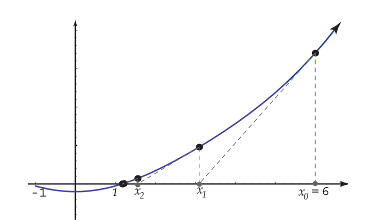


Figura 4.17 $f(x) = x^2 - \cos(x) - 1$

c) Sucesión Periódica (cíclica). $x^3 - x - 3 = 0$ con $x_0 = 0$

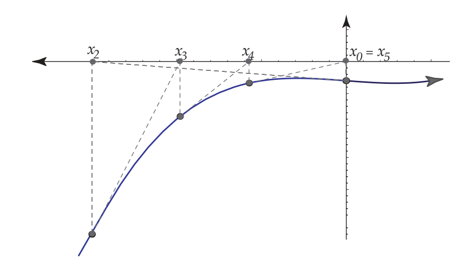


Figura 4.18 $f(x) = x^3 - x - 3$

d) (Convergencia lenta). $(x-1)^3 = 0$ con $x_0 = -2$

4.36 Sea $f(x) = x^3 - 2\cos(x) - 3$. Según el teorema de Taylor

$$f(\alpha) = f(x) + f'(x)(\alpha - x) + \frac{1}{2}(\alpha - x)^2 f''(\xi), \quad \text{con} \quad \xi \quad \text{entre} \quad x \ \text{y} \ \alpha$$

Si $\alpha = \pi/4$ y $x = 1$, calcule ξ.

4.37 Sea f una función dos veces derivable en un entorno I de una de sus raíces x^*. Si $x_n \in I$, la expansión de Taylor de orden dos para f, alrededor de $x = x_n$, es

$$f(x^*) = f(x_n) + f'(x_n)(x^* - x_n) + \frac{f''(\xi)}{2}(x^* - x_n)^2$$

Suponiendo que el *resto* es despreciable, despeje x^* del factor lineal y obtenga la fórmula de Newton.

4.38 Considere la función $f(x) = \text{sgn}(x-1)\sqrt{|x-1|}$ donde $\text{sgn}(u) = \begin{cases} 1 & \text{si } u > 0 \\ 0 & \text{si } u = 0 \\ -1 & \text{si } u < 0 \end{cases}$ El gráfico de la función es

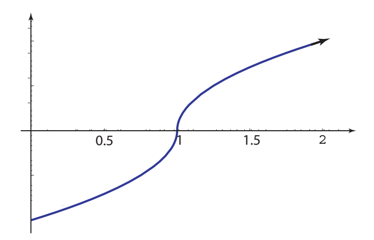


Figura 4.19 $f(x) = \text{sgn}(x-1)\sqrt{|x-1|}$

Claramente $x^* = 1$ es un cero de f. Use el método de Newton e intente aproximar este cero. Comente el resultado de sus experimentos.

4.12 Método de Newton: Teorema de Convergencia. Orden de Convergencia.

Se pueden establecer condiciones *suficientes* sobre f, f' y f'' para garantizar que el método de Newton converge. No siempre es fácil verificar estas condiciones excepto para algún tipo especial de funciones. En todo caso, si no se cumplen las hipótesis de estos teoremas, todavía puede pasar que haya convergencia pues son teoremas que dan *condiciones suficientes* de convergencia.

El primer teorema establece que si la aproximación inicial está *"suficientemente cercana"* a un cero x^* de f entonces el método de Newton converge a x^*.

¿Qué significa la frase "si la aproximación inicial está *suficientemente cercana* a un cero x^*"?. Hablando muy generalmente, si $|f'(x)| \geq m > 0$ y $|f''(x)| \leq M$ en un intervalo I centrado en x^*, entonces aplicando la fórmula de Taylor se obtiene

$$|x_n - x^*| \leq \left(\frac{M|x_0 - x^*|}{2m}\right)^n |x_0 - x^*| \tag{4.9}$$

Para conseguir la convergencia es suficiente con escoger x_0 y el intervalo I de tal manera que

$$\frac{M|x_0 - x^*|}{2m} < 1.$$

Con las condiciones del teorema de convergencia es posible manipular I dándole un tamaño adecuado (tan pequeño como sea necesario) de tal manera que si tomamos x_0 en este intervalo, obtengamos convergencia.

Los razonamientos que siguen están basados en el teorema de Taylor. Si x^* es un cero de f en un intervalo $[a,b]$ la expansión de Taylor para f alrededor de $u = x_n$, en x^* es

$$0 = f(x^*) = f(x_n) + f'(x_n)(x^* - x_n) + \frac{1}{2}(x^* - x_n)^2 f''(\xi_n), \quad \text{con} \quad \xi_n \quad \text{entre} \quad x_n \text{ y } \quad x^*$$

es decir

$$0 = f(x_n) + f'(x_n)(x^* - x_n) + \frac{1}{2}(x^* - x_n)^2 f''(\xi_n), \quad \text{con} \quad \xi_n \quad \text{entre} \quad x_n \text{ y } \quad x^*$$

El primer teorema establece la naturaleza local de la convergencia del método de Newton en el caso de ceros simples, es decir ceros x^* para las que $f(x^*) \neq 0$.

Teorema 4.8

Sea x^* un cero simple de f y sea $I_\varepsilon = \{x \in \mathbb{R} : |x - x^*| \leq \varepsilon\}$. Asumamos que $f \in C^2(I_\varepsilon)$ y que ε es lo suficientemente pequeño de tal manera que para este intervalo existen $m, M > 0$ tal que $|f'(x)| \geq m > 0$ y $|f''(x)| \leq M$. Si en el método de Newton, algún x_j satisface

$$x_j \in I_\varepsilon, \quad \text{y} \quad |e_j| = |x_j - x^*| < 2\frac{m}{M} \tag{4.10}$$

entonces $\lim_{n \longrightarrow \infty} x_n = x^*$

Prueba. Supongamos que estamos en la situación (4.10). Sea $\alpha = \frac{1}{2} \cdot \frac{M}{m}|e_j| < 1$. La expansión de Taylor para f alrededor de $u = x_j$, en x^* es

$$0 = f(x^*) = f(x_j) + f'(x_j)(x^* - x_j) + \frac{1}{2}(x^* - x_j)^2 f''(\xi_j), \quad \text{con} \quad \xi_j \quad \text{entre} \quad x_j \text{ y } \quad x^*$$

De aquí podemos despejar $e_{j+1} = |x_{j+1} - x^*|$ en términos de $e_j = |x_n - x^*|$

$$\begin{aligned} -f(x_j) - f'(x_j)(x^* - x_j) &= \tfrac{1}{2}(x^* - x_j)^2 f''(\xi_j) \\ x_j - \frac{f(x_j)}{f'(x_j)} - x^* &= \tfrac{1}{2}(x^* - x_j)^2 \frac{f''(\xi_j)}{f'(x_j)} \\ x_{j+1} - x^* &= \tfrac{1}{2}(x^* - x_j)^2 \frac{f''(\xi_j)}{f'(x_j)} \\ e_{j+1} &= \tfrac{1}{2} e_j^2 \frac{f''(\xi_j)}{f'(x_j)} \end{aligned}$$

entonces

$$|e_{j+1}| \le \frac{1}{2} \cdot \frac{M}{m} |e_j|^2 \;=\; \alpha\, |e_j| \tag{4.11}$$

Ahora, como $|e_j|^2 < 4\dfrac{m^2}{M^2}$,

$$x_{j+1} \in I_\varepsilon, \quad \text{y} \quad |e_{j+1}| = |x_{j+1} - x^*| < 2\frac{m}{M}$$

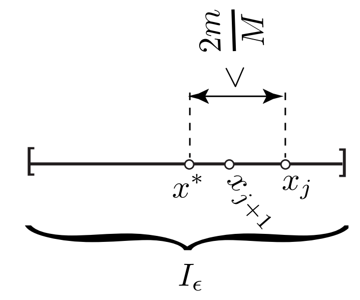


Figura 4.20 $x_{j+1} \in I_\varepsilon$

Ahora podemos repetir el razonamiento con x_{j+1} en vez de x_j (ya que de nuevo estamos en la situación (4.10)) y concluir, por inducción, que $|e_{j+k}| \le \alpha^k |e_j|$.

Como $\alpha < 1$, $\lim\limits_{k \longrightarrow \infty} e_k = 0$, es decir, $\lim\limits_{k \longrightarrow \infty} x_k = x^*$.

Ejemplo 4.19

Considere la ecuación $x^2 - A = 0$ con $A > 0$. Esta ecuación se usa para aproximar $\sqrt{A}$ con el método de Newton. Podemos usar el teorema (4.8) para demostrar que el esquema iterativo de Newton converge si x_0 se toma "suficientemente cercano" a $\sqrt{A}$. En efecto,

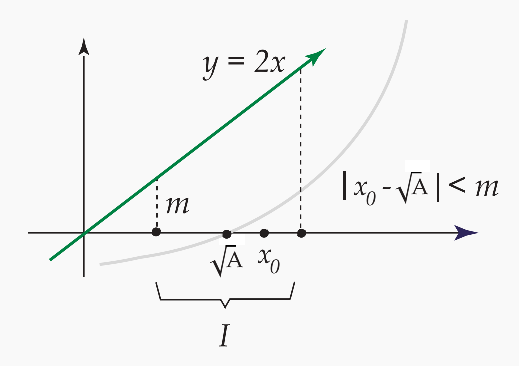


Figura 4.21 x_0 está "suficientemente cercano" a $\sqrt{A}$.

Como $f'(x) = 2x$ y $f''(x) = 2 = M$ basta tomar un intervalo I con centro en $\sqrt{A}$ en el que $f'(x) > 0$, digamos $I_\varepsilon = [\sqrt{A} - \varepsilon, \sqrt{A} + \varepsilon]$ con $\sqrt{A} - \varepsilon > 0$ y entonces el mínimo absoluto de f' en I_ε es $m = 2(\sqrt{A} - \varepsilon) > 0$ (ver figura **??**).

Si tomamos x_0 en I tal que $|x_0 - \sqrt{A}| < m$ entonces

$$x_0 \in I_\varepsilon \quad \text{y} \quad |x_0 - \sqrt{A}| < 2 \cdot \frac{m}{M} = m$$

con lo cual tendríamos convergencia.

En realidad, en este caso, el método de Newton converge para cualquier $x_0 > 0$. Para demostrar esto, se podría usar en algún momento el teorema (4.8) pero es innecesario: La convergencia se puede establecer sin el teorema (ver ejercicio 4.41).

Ejemplo 4.20

Consideremos la ecuación $(x-1)^2 = 0$. En este caso, el cero $x^* = 1$ no es un cero simple de $f(x) = (x-1)^2$. Como $f'(x) = (x-1)$, de hecho no podríamos establecer la desigualdad $|f'(x)| \geq m > 0$ en algún intervalo I centrado en $x^* = 1$ (el mínimo absoluto de $|f'|$ es cero en estos intervalos).

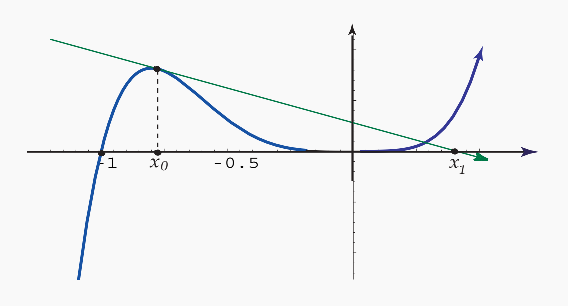
Aún así el, método de Newton converge para cualquier elección de x_0. En efecto, el esquema iterativo para esta ecuación es

$$x_{n+1} = x_n - \frac{(x_n-1)^2}{2(x_n-1)} = \frac{x_n+1}{2}$$

Luego $|x_{n+1} - 1| = \frac{1}{2}|x_n - 1|$ y en general $|x_{n+1} - 1| = \left(\frac{1}{2}\right)^{n+1} |x_0 - 1|$, de donde se sigue la convergencia si $n \to \infty$.

Ejemplo 4.21

La ecuación $x^5 + x^4 = 0$ tiene la raíz simple $x^* = -1$. La aproximación inicial $x_0 = -0.77$ no es una elección suficientemente cercana a $x^* = -1$. Con esta elección el método de Newton converge, pero no a $x^* = -1$.



EJERCICIOS

4.39 Establezca un intervalo I centrado en $x^* = \sqrt{2}$ tal que, a la luz del teorema (4.41), se pueda garantizar la convergencia del método de Newton aplicado a $x^2 - 2 = 0$ si $x_0 \in I$.

4.40 La ecuación $x^3 - 1 = 0$. tiene el cero simple $x^* = 1$. Determine $\varepsilon > 0$ de tal manera que el método de la secante converja para cualquier $x_0, x_1 \in I_\varepsilon$, $x_0 \neq x_1$ (a la luz del del teorema (4.8)).

4.41 Vamos a establecer que el esquema iterativo $x_n = 0.5\left(x_{n-1} + \frac{A}{x_{n-1}}\right)$ converge a $\sqrt{A}$, $A > 0$, para cualquier $x_0 > 0$.

- **a)** Muestre que $x_n \geq \sqrt{A}$ para todo $n > 0$. **Ayuda:** desarrolle $(x_n - \sqrt{A})^2 \geq 0$.
- **b)** Muestre que la sucesión $\{x_n\}$ definida por el esquema iterativo es decreciente a partir de x_1, i.e. $x_{n+1} \leq x_n$ si $n \geq 1$. **Ayuda:** $x_n \geq \sqrt{A}$ para todo $n > 0$.
- **c)** Muestre que la sucesión $\{x_n\}$ definida por el esquema iterativo es convergente.
- **d)** Sea $L = \lim_{n\to\infty} x_n$. Use el hecho de que $\lim_{n\to\infty} x_n = \lim_{n\to\infty} x_{n+1}$ para establecer que $L = \sqrt{A}$.

4.42 Vamos a establecer, usando otra manera de razonar, que el esquema iterativo $x_n = 0.5\left(x_{n-1} + \frac{A}{x_{n-1}}\right)$ converge a $\sqrt{A}$, $A > 0$, para cualquier $x_0 > 0$. Sea $y_n = \dfrac{x_n - \sqrt{A}}{x_n + \sqrt{A}}$.

- **a)** Muestre que $y_{n+1} = y_n^2$
- **b)** Muestre que $y_n = y_0^{2^n}$
- **c)** Muestre que, $\forall x_0 > 0$, $\lim_{n\to\infty} y_n = 0$
- **d)** Muestre que, $\forall x_0 > 0$, $\lim_{n\to\infty} x_n = \sqrt{A}$. **Ayuda:** Una forma es despejando x_n en términos de y_n y tomando el límite.

4.43 La ecuación $\operatorname{sen}(x) - 1 = 0$. tiene el cero simple $x^* = \pi/2$. ¿Se puede encontrar un intervalo I con centro en x^* y un x_0 en este intervalo de tal manera que se cumplan las condiciones del teorema (4.8)?.

4.44 La ecuación $x^5 + x^4 = 0$ tiene la raíz simple $x^* = -1$. Encuentre un intervalo I con centro en x^* y un x_0 en este intervalo de tal manera que se cumplan las condiciones del teorema (4.8).

Otros Teoremas de Convergencia. Aquí vamos a presentar otros teoremas de convergencia que son aplicables a ciertas clases especiales de funciones.

Teorema 4.9

Supongamos que el intervalo $[a,b]$ es lo suficientemente pequeño de tal manera que se cumplan las siguientes tres condiciones

a.) f tiene un único cero x^* en $[a,b]$

b.) $f(a)f(b) < 0$

c.) f' y f'' son no nulas y conservan el signo en $[a,b]$

Luego, si $x_0 \in [a,b]$, $f(x_0)f''(x_0) > 0$ y $x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)}$, entonces x_n converge a x^*.

Prueba. Como $f(a)$ y $f(b)$ deben tener signo contrario, supongamos (sin pérdida de generalidad) que $f(a) < 0$ y $f(b) > 0$ y que $f'(x) > 0$ y $f''(x) > 0$ en $[a,b]$. Entonces $f(x_0) > 0$ y $x_0 > x^*$.

Vamos a mostrar que $x_n > x^*$ y que $f(x_n) > 0$. En efecto, por la teorema de Taylor tendremos

$$0 = f(x_n) + f'(x_n)(x^* - x_n) + \frac{1}{2}(x^* - x_n)^2 f''(\xi_n), \quad \text{con } \xi_n \text{ entre } x_n \text{ y } x^*$$

y como $f''(x) > 0$ entonces

$$f(x_n) + f'(x_n)(x^* - x_n) < 0 \implies x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} > x^*$$

Luego $x_n > x^*$, $n = 0,1,2,\cdots$

Además, por los signos de f y f', se deduce que $x_n > x_{n+1}$ $(n = 0,1,2,...)$ por lo que, la sucesión $x_0, x_1, x_2, ...$ es acotada y decreciente y por lo tanto converge. El límite debe ser x^* pues si fuera $\overline{x}$ entonces,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \implies \overline{x} = \overline{x} - \frac{f(\overline{x})}{f'(\overline{x})},$$

tomando límites a ambos lados en la igualdad de la izquierda. Es decir, $f(\overline{x}) = 0$, luego $\overline{x} = x^*$ por la condición **a.)**

Teorema 4.10

Supongamos que $f(a)f(b) < 0$, $f''(x)$ es continua y $f'(x) \cdot f''(x) \neq 0$, en $[a,b]$. Entonces, si

$$\left|\frac{f(a)}{f'(a)}\right| < b-a \quad \text{y} \quad \left|\frac{f(b)}{f'(b)}\right| < b-a$$

el método de Newton converge para cualquier $x_0 \in [a,b]$.

Ejemplo 4.22

Sea $f(x) = x^3 + x + 1$, $a = -1$ y $b = -0.5$. Entonces f cumple las hipótesis del teorema (4.10),

a.) $f(a)f(b) = -0.375 < 0$

b.) $f'(x) = 3x^2 + 1$ y $f''(x) = 6x$ son continuas y no se anulan en $[-1, -0.5]$

c.) $\left|\frac{f'(a)}{f'(a)}\right| = 1/4 < b - a = 0.5$

d.) $\left|\frac{f'(b)}{f'(b)}\right| = 1.214... < b - a = 0.5$

Por lo tanto, el método de Newton converge al cero de f usando cualquier $x_0 \in [-1, -0.5]$.

Teorema 4.11

Supongamos que $f(a)f(b) < 0$, $f''(x)$ es continua y $f'(x) \cdot f''(x) \neq 0$, en $[a,b]$. Entonces, si para un $x_0 \in [a,b]$ se cumple

$$f(x_0)f''(x_0) > 0$$

el método de Newton converge (monotonamente) a un $\alpha \in [a,b]$.

Ejemplo 4.23

Sea $f(x) = x^3 + x + 1$, $a = -1$ y $b = -0.5$. Entonces f cumple las hipótesis del teorema 16,

a.) $f(a)f(b) = -0.375 < 0$

b.) $f'(x) = 3x^2 + 1$ y $f''(x) = 6x$ son continuas y no se anulan en $[-1, -0.5]$

c.) Sea $x_0 = -0.75 \in [-1, -0.5]$. $f(x_0)f''(x_0) = 0.77343... > 0$

Por lo tanto, el método de Newton converge a una raíz $\alpha \in [-1, -0.5]$.

4.13 Método de Newton: Estimación del error

Primero vamos a establecer un resultado que nos da una estimación muy general del error para luego hacer un refinamiento más manejable.

Teorema 4.12

Si x_k es una aproximación del un cero x^* de f tal que ambos están en el mismo intervalo $[a, b]$ y si $|f'(x)| \geq m_1 > 0$ en este intervalo, entonces

$$|x_k - x^*| \leq \frac{|f(x_k)|}{m_1}$$

Prueba. Aplicando el teorema de valor medio

$$f(x_k) - f(x^*) = (x_k - x^*)f'(\xi)$$

con ξ entre x_k y x^*, así $\xi \in [a, b]$. Luego, como $f(x^*) = 0$ y $|f'(\xi)| \geq m_1$, tenemos

$$|f(x_k)| \geq m_1|x_k - x^*|$$

de donde se obtiene el resultado.

Teorema 4.13

Si el método de Newton converge al cero x^* de f en $[a,b]$ y si en este intervalo $|f'(x)| \geq m > 0$ y $|f''(x)| < M$, entonces para n suficientemente grande

$$|x_n - x^*| \leq \frac{M}{2m}(x_n - x_{n-1})^2 \leq |x_n - x_{n-1}|$$

Prueba. Por el teorema de Taylor

$$f(x_n) = f(x_{n-1}) + f'(x_{n-1})(x_n - x_{n-1}) + \frac{1}{2}(x_n - x_{n-1})^2 f''(\xi_{n-1}), \quad \text{con } \xi_{n-1} \text{ entre } x_{n-1} \text{ y } x_n$$

Ahora, $f(x_{n-1}) + f'(x_{n-1})(x_n - x_{n-1}) = 0$, entonces

$$|f(x_n)| \leq \frac{1}{2}M(x_n - x_{n-1})^2$$

de donde, por el teorema (4.12),

$$|x^* - x_n| \leq \frac{M}{2m}(x_n - x_{n-1})^2$$

Así, si hay convergencia, $|x_n - x_{n-1}| \longrightarrow 0$ si $n \longrightarrow \infty$. Por tanto, si n es suficientemente grande

$$\frac{M}{2m}(x_n - x_{n-1})^2 \leq |x_n - x_{n-1}|$$

con lo que se obtiene el resultado.

Por supuesto, sería mejor usar $\frac{M}{2m}(x_n - x_{n-1})^2$ como cota de error.

Orden de Convergencia para ceros Simples.

Teorema 4.14

Supongamos que el método de Newton converge a un cero simple x^* de f Si f' y f'' existen y son continuas en los alrededores de x^* y $f''(x^*) \neq 0$, entonces el orden de convergencia del método de Newton es *como mínimo* dos.

Prueba.

Para establecer el orden de convergencia (para raíces simples) del método de Newton, podemos usar el polinomio de Taylor de la siguiente manera.

Ejemplo 4.24

La ecuación $x^2 = 2\cos(x) - 1$ tiene la solución (con 16 decimales exactos) $x^* = 0.7146210577792836$. Aplicando Newton con $x_0 = 0.5$ se obtiene la siguiente tabla

n	$\lvert x_n - x^*\rvert$	$\frac{M}{2m}(x_n - x_{n-1})^2$	$\lvert x_n - x_{n-1}\rvert$
1	0.04326742065505729	0.24460444363612954	0.25788847843434093
2	0.0011243785942077755	0.006532090995081504	0.04214304206084951
3	$8.085458267359513 \times 10^{-7}$	$4.64302230230964 \times 10^{-6}$	0.0011235700483810396
	$\vdots$	$\vdots$	

Tabla 4.1

Sea x^* es una raíz simple de f y supongamos que la iteración de Newton converge a x^*. Entonces podemos calcular la expansión de Taylor para f alrededor de $u = x_n$

$$0 = f(x^*) = f(x_n) + f'(x_n)(x^* - x_n) + \frac{1}{2}(x^* - x_n)^2 f''(\xi_n), \quad \text{con} \quad \xi_n \quad \text{entre} \quad x_n \;\; \text{y} \;\; x^*$$

Como tenemos la convergencia asegurada, $\xi_n \longrightarrow x^*$ conforme $x_n \longrightarrow x^*$.

De aquí podemos despejar $e_{n+1} = |x_{n+1} - x^*|$ en términos de $e_n = |x_n - x^*|$ (lo que nos permitirá analizar la velocidad de convergencia) de la siguiente manera: como $f'(x^*) \neq 0$ y tenemos convergencia, supongamos que los x_n que consideramos ya están en un entorno alrededor de x^* en el que $f'(x_n) \neq 0$.

$$\begin{aligned}
-f(x_n) - f'(x_n)(x^* - x_n) &= \tfrac{1}{2}(x^* - x_n)^2 f''(\xi_n) \\
x_n - \frac{f(x_n)}{f'(x_n)} - x^* &= \tfrac{1}{2}(x^* - x_n)^2 \frac{f''(\xi_n)}{f'(x_n)} \\
x_{n+1} - x^* &= \tfrac{1}{2}(x^* - x_n)^2 \frac{f''(\xi_n)}{f'(x_n)} \\
e_{n+1} &= \tfrac{1}{2}\, e_n^2 \frac{f''(\xi_n)}{f'(x_n)}
\end{aligned}$$

Como $f'(x^*) \neq 0$ se sigue que

$$\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|^2} = K = \left|\frac{1}{2}\frac{f''(x^*)}{f'(x^*)}\right|$$

Si $f''(x^*) \neq 0$, entonces $K > 0$ y la convergencia es cuadrática.

Orden de Convergencia para ceros de Multiplicidad $m > 1$.

Definición 4.2 (Ceros múltiples).

Supongamos que f y sus derivadas $f', f'', \cdots, f^{(n)}$ existen y son continuas en un intervalo I que contiene a x^* con $f(x^*) = 0$. Diremos que el cero x^* es de orden n si

$$f(x^*) = 0,\ f'(x^*) = 0,\ \cdots, f^{(n-1)}(x^*) = 0 \ \text{ pero } \ f^{(n)}(x^*) \neq 0$$

Un cero x^* es de orden 1 si $f(x^*) = 0$, $f'(x^*) \neq 0$. En este caso se dice que x^* es un *cero simple.*

El método de Newton solo exhibe convergencia cuadrática para ceros simples. En el caso de multiplicidad 2 la convergencia es lineal con $K = 1/2$: En efecto, si x^* es un cero de multiplicidad $m = 2$, es decir, si $f'(x^*) = 0$ y $f''(x^*) \neq 0$, de acuerdo a la fórmula de Taylor tendremos (como vimos antes)

$$e_{n+1} = \frac{1}{2} e_n^2 \frac{f''(\xi_n)}{f'(x_n)}$$

Ahora, usando el teorema del valor medio para derivadas, $\dfrac{f'(x^*) - f'(x_n)}{x^* - x_n} = f''(\eta)$ con η entre x^* y x_n. Entonces

$$\begin{aligned}
e_{n+1} &= \tfrac{1}{2} e_n^2 \frac{f''(\xi_n)}{f'(x_n)} \\
&= \tfrac{1}{2} e_n^2 \frac{f''(\xi_n)}{f'(x^*) + e_n f''(\eta)} \\
&= \tfrac{1}{2} e_n^2 \frac{f''(\xi_n)}{e_n f''(\eta)} \\
&= \tfrac{1}{2} e_n \frac{f''(\xi_n)}{f''(\eta)}
\end{aligned}$$

Y así

$$\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|} = \frac{1}{2}\frac{f''(x^*)}{f''(x^*)} = 1/2$$

En general, si $m>1$, $e_{n+1}=e_n-e_n/m$. Esto indica que si la multiplicidad del cero es $m>1$, entonces el método de Newton converge solo linealmente con tasa $K=1-1/m$.

Por ejemplo si $m=20$, se necesitarían de 45 iteraciones para ganarse un dígito decimal adicional (más lento que bisección).

Para restaurar la convergencia en el caso de ceros múltiples podemos hacer un par de modificaciones,

- (1870, E.Schröder) Si la multiplicidad m de un cero es conocida a priori entonces

$$x_{n+1}=x_n-m\frac{f(x_n)}{f'(x_n)}$$

  converge al menos cuadráticamente (ejercicio 4.47).

- En todo caso, si se desconoce la multiplicidad de un cero x^*, al menos este cero es un cero simple de $u(x)=f(x)/f'(x)$, por lo que el método de Newton convergerá cuadráticamente para esta función (ejercicio 4.48).

  En este caso, la iteración de Newton es

$$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)-\dfrac{f(x_n)f''(x_n)}{f'(x_n)}}$$

Esta fórmula recursiva requiere no solo f' sino también f'' por lo que el costo computacional aumenta. Hay que tener en cuenta además que aunque teóricamente la convergencia es cuadrática, para cierto tipo de funciones, por ejemplo polinomios con raíces múltiples, hay una barrera que no permite acercarse a la raíz con la precisión esperada.

4.14 Métodos de Orden Cúbico. Método de Euler.

El método de Newton puede obtenerse con un polinomio de Taylor de orden 1. Si aumentamos el orden del polinomio de Taylor , se pueden obtener métodos de orden superior. Aunque, para ceros simples hay métodos modificados que alcanzan orden de convergencia alto [15], aquí solo discutiremos brevemente un método de orden cúbico, llamado *método de Euler*.

Usando el polinomio de Taylor de orden 2,

$$f(x_n+h)=f(x_n)+hf'(x_n)+\frac{h^2}{2}f''(x_n)+O(h^3)$$

si $f(x_n+h)=0$

$$0\approx f(x_n)+h_nf'(x_n)+\frac{h_n^2}{2}f''(x_n)$$

Si f' y f'' no se anulan y si $f'(x_n)^2 \geq 2f(x_n)f''(x_n)$, se puede resolver para h_n (tomando el signo " $-$ ")

$$h_n \approx -\frac{f'(x_n)}{f''(x_n)}\left(1-\sqrt{1-\frac{2f(x_n)f'(x_n)}{(f'(x_n))^2}}\right)$$

Reordenando

$$x_{n+1} = x_n - u(x_n)\,\frac{2}{1+\sqrt{1-2t(x_n)}}$$

con $u(x) = \dfrac{f(x)}{f'(x)}$ y $t(x) = u(x)\,\dfrac{f''(x)}{f'(x)}$

Ejemplo 4.25

Sea $f(x) = x^2 - 2\cos(x) + 1$. Un cero de esta función es $x^* = 0.7146210577$ (con 10 decimales exactos.) Aplicando el método de Newton y el método de Euler se obtiene la siguiente tabla

x_n	Método de Newton	Método de Euler
x_0	0.5	0.5
x_1	0.7578884784343409	0.7139946504857623
x_2	0.7157454363734914	0.7146210577596979
x_3	0.7146218663251104	0.7146210577792835

Tabla 4.2

Para obtener 10 dígitos exactos, nos ganamos una iteración con el método de Euler, pero con un costo computacional más alto.

EJERCICIOS

4.45 Muestre que si $m > 0,\ M > 0$ y $|x_n - x_{n-1}| \longrightarrow 0$ si $n \longrightarrow \infty$, entonces si n es suficientemente grande,

$$\frac{M}{2m}(x_n - x_{n-1})^2 \leq |x_n - x_{n-1}|$$

4.46 El esquema iterativo $x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)}$ se puede ver como un esquema iterativo de un problema de punto fijo. Use el teorema (4.6) para establecer, bajo hipótesis adecuadas, que el orden de convergencia del método de Newton es al menos dos.

4.47 Vamos a establecer que si la multiplicidad m de un cero x^* es conocida a priori entonces si $x_{n+1} = x_n - m\dfrac{f(x_n)}{f'(x_n)}$ converge, la convergencia es cuadrática.

a) Muestre que $x^*-x_{n+1}=x^*-x_n+m\dfrac{f(x_n)}{f'(x_n)} \Longrightarrow (x^*-x_{n+1})f'(x_n)=H(x_n)$ donde $H(x)=(x^*-x)f'(x)+mf(x)$.

b) Muestre que $H^{(k)}(x)=mf^{(k)}(x)+(x^*-x)f^{(k+1)}(x)-kf^{(k)}(x)$

c) Muestre que $H^{(m+1)}(x^*)\neq 0$.

d) Aplique el teorema de Taylor a H para mostrar que $H(x)=\dfrac{(x-x^*)^{m+1}}{(m+1)!}H^{(m+1)}(\varepsilon_1)$

e) Aplique el teorema de Taylor a f' para mostrar que, como x^* es cero de multiplicidad m de f, entonces $f'(x)=\dfrac{(x-x^*)^{m-1}}{(m-1)!}f^{(m)}(\varepsilon_2)$

f) Use (4.47.a) para mostrar que $\dfrac{x^*-x_{n+1}}{(x^*-x_n)^2}=\dfrac{H^{(m+1)}(\varepsilon_1)}{m(m+1)f^{(m)}(\varepsilon_2)}$

g) Concluya: ¿porqué el orden de convergencia es al menos dos?

4.48 Vamos a establecer que si x^* es un cero de f de multiplicidad m, entonces x^* es un cero simple de $u(x)=f(x)/f'(x)$ y si $x_{n+1}=x_n-u(x_n)/u'(x_n)$ converge, la convergencia es cuadrática.

Para probar esto necesitamos la siguiente definición: Un cero x^* de f es un cero de multiplicidad m si $f(x)=(x-x^*)^m Q(x)$ donde $\lim_{x\to x^*} Q(x)\neq 0$.

a) Muestre que si x^* es cero de f, $u(x^*)=0$

b) Muestre que $u'(x^*)=0$

4.49 Verifique que para $(x-1)^2=0$, el método de Newton converge. Además verifique que la convergencia es lineal. Modifique el método de Newton para obtener convergencia cuadrática.

4.50 Considere la ecuación $(x-0.123456789)^3=0$.

a) ¿Qué multiplicidad tiene la raíz $x=0.123456789$?

b) Aplique Newton con $x_0=1$

c) Modifique la ecuación de tal manera que el orden de convergencia sea cuadrático. Hacer esto de dos maneras: Usando la multiplicidad de la raíz y usando la ecuación $u(x)=f(x)/f'(x)$

4.51 Considere la ecuación $4(x-1)^3\cos(x)=0$.

a) ¿Qué multiplicidad tiene la raíz $x=1$?

b) Aplique Newton con $x_0=0.5$

c) Modifique la ecuación de tal manera que el orden de convergencia sea cuadrático. Hacer esto de dos maneras: Usando la multiplicidad de la raíz y usando la ecuación $u(x)=f(x)/f'(x)$

4.52 $x=2$ es un cero, de multiplicidad $m=7$, del polinomio $P(x)=-1536+6272x-11328x^2+11872x^3-7952x^4+3528x^5-1036x^6+194x^7-21x^8+x^9$. Repita la parte *c.)* del ejercicio anterior con este polinomio.

4.53 $x=1$ es un cero, de multiplicidad $m=7$, del polinomio $P(x)=2-19x+81x^2-204x^3+336x^4-378x^5+294x^6-156x^7+54x^8-11x^9+x^{10}$. Repita Repita la parte *c.)* del ejercicio anteriorcon este polinomio.

4.54 Implemente el método de Euler con VBA Excel.

4.55 Sea $x^2-2\cos(x)+1=0$ Aplique el método de Euler y el método de Newton para resolver esta ecuación con $x_0=0.1$. Compare resultados.

4.56 $x=2$ es un cero, de multiplicidad $m=7$, del polinomio $P(x)=-1536+6272x-11328x^2+11872x^3-7952x^4+3528x^5-1036x^6+194x^7-21x^8+x^9$. Aplique el método de Euler a este polinomio con $x_0=2.2$ (si usa Excel,usar Xnumbers). Analice el resultado.

4.57 Sea $g(x)=x-\dfrac{(x^2-A)(5x^2-A)}{8x^3}$.

a) Verifique que $\sqrt{A}$ es un punto fijo de g

b) Si la iteración $x_{n+1} = g(x_n)$ converge, muestre que la convergencia es de orden $q = 3$.

4.15 Un método híbrido: Newton-Bisección.

Si el método de Newton falla (en algún sentido) en una iteración, podemos usar bisección para dar un pequeño salto y regresar al método de Newton lo más pronto posible.

Supongamos que $f(a)f(b) < 0$. Sea $x_0 = a$ o $x_0 = b$. En cada iteración una nueva aproximación x' es calculada y a y b son actualizados como sigue

a.) si $x' = x_0 - \dfrac{f(x_0)}{f'(x_0)}$ cae en $[a,b]$ lo aceptamos, sino usamos bisección, es decir $x' = \dfrac{a+b}{2}$.

b.) Actualizar: $a' = x',\ b' = b$ o $a' = a,\ b' = x'$, de tal manera que $f(a')f(b') \leq 0$.

Para garantizar que $x' = x_0 - \dfrac{f(x_0)}{f'(x_0)} \in [a,b]$, no debemos usar directamente este cálculo para evitar la división por $f'(x_0)$ (que podría causar problemas de "overflow" o división por cero). Mejor usamos un par de desigualdades equivalentes.

Observemos que $x_{n+1} \in]a_n, b_n[$ si y sólo si

$$a_n < x_n - \frac{f(x_n)}{f'(x_n)} < b_n$$

entonces

- Si $f'(x_n) > 0$

$$(a_n - x_n)f'(x_n) < -f(x_n) \ \text{ y } \ (b_n - x_n)f'(x_n) > -f(x_n)$$

- Si $f'(x) < 0$

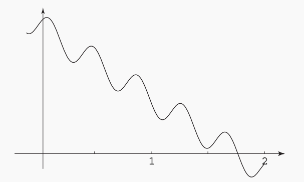
$$(a_n - x_n)f'(x_n) > -f(x_n) \ \text{ y } \ (b_n - x_n)f'(x_n) < -f(x_n)$$

En este algoritmo, se pasa a bisección si $x' = x_0 - \dfrac{f(x_0)}{f'(x_0)}$ sale del intervalo, pero esto no indica necesariamente que la iteración de Newton tenga algún tipo de problema en ese paso. Lo que si es importante es que se podría escoger un intervalo $[a,b]$ hasta con extremos relativos (que podrían ser mortales para el método de Newton) y el tránsito sería seguro de todas formas.

Un algoritmo similar aparece en [12] (pág. 366). En la implementación, se pasa a bisección si x' sale del intervalo o si la reducción del intervalo no es suficientemente rápida.

Ejemplo 4.26

Consideremos la función $f(x) = 0.2\,\text{sen}(16x) - x + 1.75$ Esta función tiene un cero en $[1,2]$. Así que $a = 1$ y $b = 2$. Iniciamos con $x_0 = 1$. La tabla (4.3) muestra el método de Newton y Híbrido Newton-bisección aplicado a esta ecuación. El método de Newton diverge mientras que el híbrido aproxima la solución adecuadamente.



n	Newton x_i	Error Estimado	Híbrido x_i	Error Estimado	Método Usado
1	1.170357381	0.170357381	1.17035738114819	0.170357381	Newton
2	0.915271273	0.255086108	1.58517869057409	0.414821309	Bisección
3	1.310513008	0.395241735	1.79258934528705	0.207410655	Bisección
4	1.539337071	0.228824064	1.76166924922784	0.030920096	Newton
5	1.476007274	0.063329797	1.76306225245136	0.001393003	Newton
6	1.565861964	0.08985469	1.76306130340890	9.49042×10^{-7}	Newton
	...		...		
50	-2873.507697				
51	-1720.319542				

Tabla 4.3 Método de Newton e híbrido Newton-bisección aplicado a $0.2\,\text{sen}(16x) - x + 1.75 = 0$ en $[1,2]$.

4.15.1 Algoritmo e Implementación en VBA Excel.

Hoja Excel para el Híbrido Newton-bisección. Para hacer una hoja Excel, usamos como referencia la figura (4.22).

	A	B	C	D	E	F	G
1							
2					Calcule		
3	f(x) =	0.2*sin(16*x)-x+1.75					
4	f'(x) =	3.2*cos(16*X)-1					
5							
6	x0 = a				Hibrido	Error	Método
7	a	b	delta	maxItr	mi = xi	Estimado	Usado
8	-1	2	0.00005	10	-0.30924504293887	0.690754957	Newton
9					0.84537747853057	1.154622521	Bisección
10					1.42268873926528	0.577311261	Bisección
11					1.47972114111601	0.057032402	Newton
12					1.59202270026853	0.112301559	Newton

Figura 4.22 Hoja Excel para el híbrido Newton-bisección.



Software: Cuadernos LibreOffice y Excel

Código VBA 4.10: Híbrido Newton-Bisección con `clsMathParser`

```
Option Explicit
Private Sub CommandButton1_Click()
Dim fx, fpx, a, b, delta, maxItr, fi, co
    fx = Cells(3, 2)
```

```
    fpx = Cells(4, 2)
    a = Cells(8, 1)
    b = Cells(8, 2)
    delta = Cells(8, 3)
    maxItr = Cells(8, 4)
    Call hibridoNB(fx, fpx, a, b, delta, maxItr, 8, 5)
End Sub

Sub hibridoNB(fx, fpx, a0, b0, delta, maxItr, fi, co)
Dim f As New clsMathParser
Dim fp As New clsMathParser
Dim test1 As Boolean, test2 As Boolean
Dim k, x0, x1, dx, okfx, okfpx, fx0, fpx0, fx1, fa, a, b

okfx = f.StoreExpression(fx)
okfpx = fp.StoreExpression(fpx)

If Not okfx Then
    MsgBox ("Error en f: " + f.ErrorDescription)
    Exit Sub
End If

If Not okfpx Then
    MsgBox ("Error en fp: " + fp.ErrorDescription)
    Exit Sub
End If
'No modificar a y b
a = a0
b = b0
If Sgn(f.Eval1(a)) = Sgn(f.Eval1(b)) Then
        MsgBox ("Error: f(a)f(b)>=0")
        Exit Sub
End If

k = 0
x0 = a
Do
    fx0 = f.Eval1(x0)
    fpx0 = fp.Eval1(x0)
test1 = (fp(x0) > 0 And (a - x0) * fp(x0) < -f(x0) And (b - x0) * fp(x0) > -f(x0))
test2 = (fp(x0) < 0 And (a - x0) * fp(x0) > -f(x0) And (b - x0) * fp(x0) < -f(x0))

If test1 Or test2 Or fx0 = 0 Then
     x1 = x0 - fx0 / fpx0
     dx = Abs(x1 - x0)
     x0 = x1
     If Sgn(f.Eval1(a)) <> Sgn(f.Eval1(x1)) Then
           b = x1
     Else: a = x1
```

```
        End If
        Cells(fi + k, co) = x1
        Cells(fi + k, co + 1) = dx
        Cells(fi + k, co + 2) = "Newton"
    Else
       x1 = a + 0.5 * (b - a)
       dx = (b - a) / 2
       x0 = x1

       If Sgn(f.Eval1(a)) <> Sgn(f.Eval1(x1)) Then
             b = x1
       Else: a = x1
       End If

       Cells(fi + k, co) = x1
       Cells(fi + k, co + 1) = dx
       Cells(fi + k, co + 2) = "Bisecci\'on"
    End If
   k = k + 1
   Loop Until dx < delta Or k > maxItr
   End Sub
```

EJERCICIOS

4.58 Afine la implementación en Excel (observe que no verifica que haya cambio de signo).

4.59 $f(x) = x^3\cos(x) - x\,\text{sen}(2x) + 1$ tiene un cero en el intervalo $[1,2]$. Pruebe aproximar el cero de f con $x_0 = 1.5$ usando Newton y pruebe usando el híbrido Newton-Bisección usando el intervalo $[1,2]$.

4.60 $f(x) = x^{20} - 1$ tiene un cero en el intervalo $[0.5,1.5]$. Pruebe aproximar el cero de f con $x_0 = 0.5$ usando Newton y pruebe usando el híbrido Newton-Bisección usando el intervalo $[0.5,1.5]$.

4.61 Implemente en VBA el algoritmo descrito en [12] (pág. 366).

4.62 Resolver $4(x-1)^3\cos(x) = 0$.

4.16 El Método de la Falsa Posición

Este método es todavía más viejo que el método de Newton, de hecho aparece en textos Indios del siglo V ([18]). Su orden de convergencia no vas allá de ser lineal. En esta sección vamos a dar una breve descripción del método.

La idea de este método es calcular la recta secante que une los puntos extremos $(a_1, f(a_1))$ y $(b_1, f(b_1))$. Luego se determina el punto m en que esta recta corta el eje x y este valor entra a jugar el papel que en el método de bisección jugaba el punto medio.

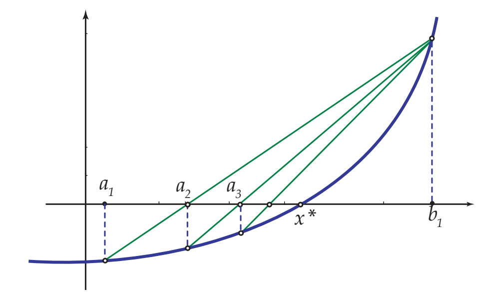


Figura 4.23 Método de la falsa posición

La recta secante tiene que une los puntos $(a_1, f(a_1))$ y $(b_1, f(b_1))$ tiene ecuación

$$y = f(a_1) + \frac{f(b_1) - f(a_1)}{a_1 - b_1}(x - a_1)$$

Al resolver $y = 0$ se despeja el valor de x, obteniendo:

$$m = a_1 - \frac{f(a1)(b_1 - a_1)}{f(b_1) - f(a_1)} \quad \Longrightarrow \quad m = \frac{a_1 f(b_1) - b_1 f(a_1)}{f(b_1) - f(a_1)}$$

En este método no se conoce a priori el número de iteraciones requeridas para alcanzar la precisión deseada así que se usa un número máximo de iteraciones.

4.16.1 Algoritmo.

El algoritmo es

Algoritmo 4.4: Método de falsa posición

Datos: $f \in C[a,b]$ con $f(a)f(b) < 0$, δ, maxItr
Salida: Una aproximación x_n de un cero de f.

$k = 0$;
$a_n = a,\ b_n = b.$;
repeat
  $x_n = \dfrac{a_n f(b_n) - b_n f(a_n)}{f(b_n) - f(a_n)}$;
  $e_1 = x_n - a_n;\ e_2 = b_n - x_n$;
  if $f(x_n)f(a_n) > 0$ **then**
    $a_n = x_n$
  else
    $b_n = x_n$
  $k = k + 1$;
until $(\text{máx}(e_1, e_2) \leq \delta \ \vee \ k \geq \text{maxItr})$;
return x_n

4.16.2 Teorema de Convergencia. Orden de Convergencia.

Dado el tratamiento breve de esta sección, vamos a establecer un teorema de convergencia para el caso sencillo en el que f es convexa. Digamos que la situación es la misma que la que se presenta en la figura (4.23). Digamos entonces que en $[a,b]$ f cumple

$$f''(x) > 0, f(a) < 0, f(b) > 0$$

entonces f tiene exactamente un cero x^* en $[a,b]$ y la secante que conecta los puntos $(a, f(a))$ y $(b, f(b))$ está encima del gráfico de f e interseca el eje X a la izquierda de x^*. Esto mismo pasará para las otras secantes, lo que significa que el punto $x = b$ permanece fijo mientras que el otro punto se actualiza en cada paso. Lo que obtenemos es una sucesión monótona

$$x_{n+1} = x_n - f(x_n) \cdot \frac{(x_n - b)}{f(x_n) - f(b)}, \ n = 1, 2, ..., \ x_1 = a \tag{4.12}$$

que es creciente. Como esta sucesión permanece acotada por x^* entonces converge.

Para establecer el orden de convergencia, restamos x^* a ambos lados de (4.12) y usamos el hecho de que $f(x^*) = 0$

$$x_{n+1} - x^* = x_n - x^* - f(x_n) \cdot \frac{(x_n - b)}{f(x_n) - f(b)}$$

dividiendo por $x_n - x^*$ obtenemos

$$\frac{x_{n+1} - x^*}{x_n - x^*} = 1 - \frac{x_n - b}{f(x_n) - f(b)} \frac{f(x_n) - f(x^*)}{x_n - x^*}.$$

Haciendo $x \longrightarrow \infty$ y usando el hecho de que $x_n \longrightarrow x^*$ se sigue que

$$\lim_{x \longrightarrow \infty} \frac{x_{n+1} - x^*}{x_n - x^*} = 1 - (b - x^*) \frac{f'(x^*)}{f(b)} = K$$

o sea, convergencia lineal (en este caso $0 < K < 1$).

EJERCICIOS

4.63 Implemente el método de falsa posición.

4.64 Explique cuál es la razón de la escogencia del criterio de parada en este algoritmo.

4.65 Aplique el método de falsa posición a la ecuación $\cos(x)\cosh(x) - 1 = 0$ con $a = 3\pi/2$ y $b = 2\pi$. Compare con el resultado que se obtiene al aplicar el método de la secante y bisección.

4.17 Método de la Secante

Aunque el método de la secante es anterior[8] al método de Newton, a veces se hace una derivación de este método basado en la iteración de Newton cambiando la derivada $f'(x_k)$ por una aproximación, lo cual puede ser muy bueno pues para algunas funciones, como las definidas por integrales o una serie, en los casos en los que la derivada no es fácil de obtener. Aquí vamos a proceder igual que antes, con una idea geométrica. El método de la secante tiene orden de convergencia de al menos $p = 1.61803$ pero en un sentido que haremos más preciso al final de esta sección, este método es más rápido que el método de Newton.

Iniciando con *dos aproximaciones iniciales* x_0 y x_1, en el paso $k+1$, x_{k+1} se calcula, usando x_k y x_{k-1}, como la intersección con el eje X de la recta (secante) que pasa por los puntos $(x_{k-1}, f(x_{k-1}))$ y $(x_k, f(x_k))$

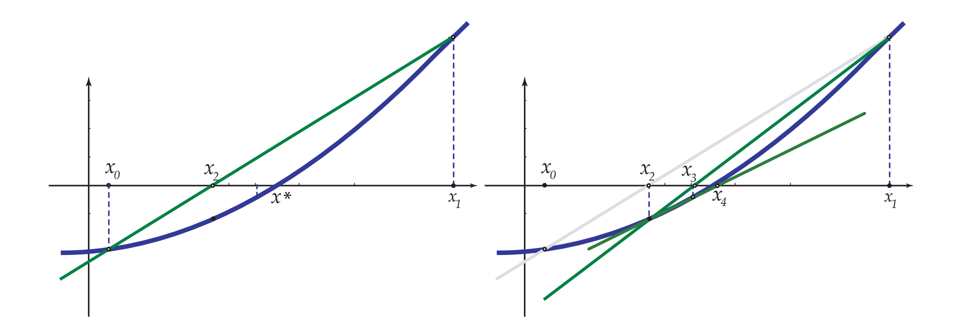


Figura 4.24 Método de la secante

Entonces, si $f(x_k) - f(x_{k-1}) \neq 0$,

$$x_{k+1} = \frac{x_{k-1} f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})}$$

Sin embargo, para cuidarnos del fenómeno de cancelación (cuando $x_k \approx x_{k-1}$ y $f(x_k)f(x_{k-1}) > 0$), rescribimos la fórmula como

$$x_{k+1} = x_k - f(x_k) \cdot \frac{(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}, \quad k \geq 1,$$

Aunque esta última versión no es totalmente segura, es al menos mejor que la anterior.

- Usualmente escogemos x_0 y x_1 de tal manera que el cero que queremos aproximar esté entre estos números. Si la función f es dos veces diferenciable en un entorno de un cero simple x^* se garantiza que si x_0 y x_1 se escogen "suficientemente cercanos" a x^*, entonces el método converge a x^*.

[8] El método de la secante ya era usando para calcular valores de la función seno en algunos textos Indios del siglo XV.

En general, si x_0 y x_1 no están suficientemente cercanos a un cero, entonces puede pasar que haya intervalos $[x_n, x_{n-1}]$ (o $[x_{n-1}, x_n]$) sin un cero en ellos. Aún así el método puede ser que converja aunque no se garantiza que sea al cero buscado.

- Cuando el método de la secante converge, $|x_k - x_{k-1}|$ eventualmente se vuelve pequeño. Pero en general, $|x_k - x_{k-1}|$ se mantiene bastante más grande que $|x_k - x^*|$. Si $|f'(x)| \geq m > 0$ en un intervalo I que contenga a x_k y a x^*, la estimación del error se podría hacer con $\dfrac{|f(x_k)|}{m}$ (ver Teorema 4.12). En todo caso, como criterio de parada podemos usar

$$|x_k - x_{k+1}| \leq \delta\,(|x_{k+1}| + 1) \text{ y } |f(x_{k+1})| < \epsilon$$

junto con un número máximo de iteraciones.

Ejemplo 4.27

Consideremos la ecuación $x^3 - 0.2x^2 - 0.2x - 1.2 = 0$. Esta ecuación tiene una solución en el intervalo $[1, 1.5]$. En realidad la solución es $x^* = 1.2$. Vamos a aplicar el método de la secante con $x_0 = 1$ y $x_1 = 1.5$.

$x_0 = 1,\ x_1 = 1.5$

Calcular x_2.

$$x_2 = x_1 - f(x_1)\cdot\frac{(x_1 - x_0)}{f(x_1) - f(x_0)} = 1.1481481481481481$$

Calcular x_3.

$$x_3 = x_2 - f(x_2)\cdot\frac{(x_2 - x_1)}{f(x_2) - f(x_1)} = 1.1875573334135374$$

Calcular x_4.

$$x_4 = x_3 - f(x_3)\cdot\frac{(x_3 - x_2)}{f(x_3) - f(x_2)} = 1.2006283753725182$$

Calcular x_5.

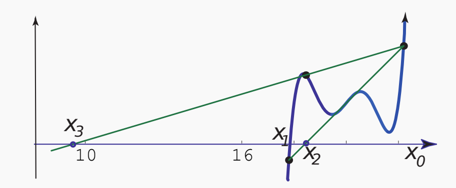
$$x_5 = x_4 - f(x_4)\cdot\frac{(x_4 - x_3)}{f(x_4) - f(x_3)} = 1.1999926413206037$$

Para estimar el error vamos a usar $|x_k - x_{k-1}|$. En la tabla (4.27) hacemos una comparación entre el error correcto para la solución $x^* = 1.2$ y la estimación del error.

k	x_k	$\lvert x_k - x^*\rvert$	$\lvert x_k - x_{k-1}\rvert$
1	1.5	0.3	0.5
2	1.14815	0.0518519	0.351852
3	1.18756	0.0124427	0.0394092
4	1.20063	0.000628375	0.013071
5	1.19999	7.35868×10^{-6}	0.000635734
6	1.200000000000030		4.31745×10^{-9}

Ejemplo 4.28

Consideremos $P(x) = x^5 - 100x^4 + 3995x^3 - 79700x^2 + 794004x - 3160075$. En la figura (4.28) se muestra la gráfica de P con las primeras dos secantes usando $x_0 = 22.2$ y $x_1 = 17$.



Ejemplo 4.28 (continuación).

P tiene un cero, $x^* = 17.846365121...$, en el intervalo $[17,22,2]$. Vamos a aproximar este cero usando $x_0 = 17$ y $x_1 = 22.2$ y también con $x_0 = 22.2$ y $x_1 = 17$.

	$x_0 = 22.2$	$x_1 = 17$		$x_0 = 22.2$	$x_1 = 17$	
k	x_{k+1}	$\lvert x_k - x_{k+1}\rvert$	$f(x_i)$	x_{k+1}	$\lvert x_k - x_{k+1}\rvert$	$f(x_i)$
1	21.70509296	4.705	1.44	21.70509296	0.494	1.446
2	21.64664772	0.058	1.36	21.63787473	0.067	1.369
3	20.61844015	1.028	6.38	20.44319288	1.194	6.354
...						
23	17.84697705	0.013	0.02	21.2200121	0.429	3.503
24	17.84635118	0.000	-0.00	21.92553159	0.705	3.475
25	17.84636513	1.39×10^{-5}	5.79283×10^{-7}	111.120	89.194	627
26	17.84636512	1.37×10^{-8}	-1.86265×10^{-9}	21.925	89.194	3.475
...				...	...	
77				20.57831656	5110.34	6.410
78				20.57831656	1.06×10^{-14}	6.410

La elección $x_0 = 22.2$ y $x_1 = 17$ muestra ser adecuada. Nos lleva a la aproximación correcta $x_{26} = 17.84636512$. En la otra elección hay un fenómeno de cuidado: La iteración 78 podría inducirnos a error pues nos presenta a $x_{78} = 20.57831656$ como un cero aproximado con error estimado 1.06×10^{-14} pero $f(x_{78}) = 6.410$!.

Notemos que hay varios intervalos (con extremos x_k, x_{k+1}) que no contienen un cero.

4.17.1 Algoritmo e Implementación en VBA Excel.

En este algoritmo, como criterio de parada usamos

$$|x_k - x_{k-1}| \leq \delta\,(|x_k| + 1) \text{ y } |f(x_k)| < \epsilon$$

junto con un número máximo de iteraciones.

Hoja Excel para el método de la secante. La implementación en VBA para Excel recibe la función f como "string", y los valores x_0, x_1, δ y maxItr. Esta implementación usa clsMathParser para leer y evaluar la función. La implementación corresponde a la hoja Excel de la figura que sigue.

Algoritmo 4.5: Método de la secante.

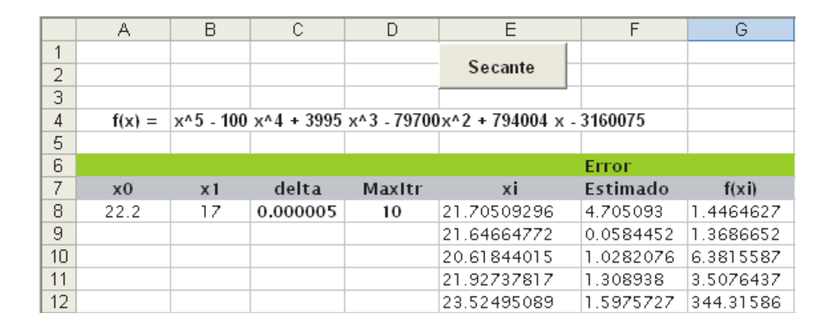
Datos: Una función continua f, las aproximaciones iniciales x_0 y x_1, δ y maxItr
Salida: Si la iteración converge a un cero, una aproximación x_k del cero.

$j = 0$;
$x_{k-1} = x_0$;
$x_k = x_1$;
while $|x_k - x_{k-1}| > \delta\,(|x_k| + 1)$ y $j < Nmax$ **do**

$$x_{k+1} = x_k - f(x_k) \cdot \frac{(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})};$$

  $x_{k-1} = x_k$;
  $x_k = x_{k+1}$;
  $j = j + 1$;

return x_{k+1};



	A	B	C	D	E	F	G
1					Secante		
2							
3							
4	f(x) =	x^5 - 100 x^4 + 3995 x^3 - 79700x^2 + 794004 x - 3160075					
5							
6						Error	
7	x0	x1	delta	MaxItr	xi	Estimado	f(xi)
8	22.2	17	0.000005	10	21.70509296	4.705093	1.4464627
9					21.64664772	0.0584452	1.3686652
10					20.61844015	1.0282076	6.3815587
11					21.92737817	1.308938	3.5076437
12					23.52495089	1.5975727	344.31586

Figura 4.25 Hoja Excel para el método de la secante

 **Software:** Cuadernos LibreOffice y Excel

Código VBA 4.11: Método de la Secante

```
Option Explicit
Private Sub CommandButton1_Click()
Dim fx, xc, xu, delta, maxItr
    fx = Cells(4, 2)
    xc = Cells(8, 1)
    xu = Cells(8, 2)
    delta = Cells(8, 3)
    maxItr = Cells(8, 4)
    Call MSecante(fx, xc, xu, delta, maxItr, 8, 5)
End Sub
Sub MSecante(fx, xc, xu, delta, maxItr, fi, co)
Dim f As New clsMathParser
Dim j, okfx, x0, x1, x2, fx1, fx0
okfx = f.StoreExpression(fx)
If Not okfx Then
    MsgBox ("Error en f: " + f.ErrorDescription)
    Exit Sub
End If
```

```
j = 0
x0 = xc
x1 = xu
Do While Abs(x0 - x1) > delta And j < maxItr
    fx1 = f.Eval1(x1)
    fx0 = f.Eval1(x0)
    x2 = x1 - fx1 * (x1 - x0) / (fx1 - fx0)
    x0 = x1
    x1 = x2
    Cells(fi + j, co) = x1
    Cells(fi + j, co + 1) = Abs(x0 - x1)
% + j, co + 2) = f.Eval1(x1)
    j = j + 1
Loop
End Sub
```

Implementación en Mathematica. Este módulo tiene un argumento `pr` opcional (con valor default = 20) para la precisión en la salida.

Código VBA 4.12: Implementación con Wolfram Mathematica.

```
MSecante[f_, xc_, xu_, delta_, maxItr_, pr_:20] :=
  Module[{j, x0, x1, x2, fx0, fx1},
        j = 1;
        x0 = xc; x1 = xu;
        fx0 = f /. {x -> x0}//N;
        While[Abs[x0 - x1] > delta && j < maxItr,
             fx1 = f /. {x -> x1};
             x2 = x1 - fx1*((x1 - x0)/(fx1 - fx0));
             fx0 = fx1; x0 = x1; x1 = x2;
             j = j + 1;
% Print[j, "    ", PaddedForm[x1, {pr, pr}],
                      "    ", PaddedForm[Abs[x0 - x1], {pr, pr}]]]]
(*Ejemplo*)
MSecante[x^3 + x + 1, -1, 2, 0.00005, 10, 30]
```

4.66 ¿Porqué, en la implementación VBA Excel, se puede hacer el siguiente cambio?

```
fx0 = f.Eval1(x0)
Do While Abs(x0 - x1) > delta And j < maxItr
    fx1 = f.Eval1(x1)
    x2  = x1 - fx1 * (x1 - x0) / (fx1 - fx0)
    fx0 = fx1
    x1  = x2
    j   = j + 1
Loop
```

4.67 Para aproximar el cero $x^* = 1$ de la función $f(x) = x^{20} - 1$,

a) Aplique el método de la secante con $x_0 = 0.5$ $x_1 = 2$. ¿Hay algún problema?.
b) Aplique el método de la secante con $x_0 = 0$ $x_1 = 1.005$.
c) Aplique el método de bisección con $x_0 = 0$ $x_1 = 3$.
d) Aplique el método de Newton con $x_0 = 0$.

4.68 Resuelva $x^3 = 0$ usando Newton con $x_0 = -0.2$. Resuelva la misma ecuación usando bisección y secante con el intervalo $[-0.2, 0.1]$

4.69 Resuelva $f = x^5 - 100 * x^4 + 3995 * x^3 - 79700 * x^2 + 794004 * x - 3160075$ usando Newton con $x_0 = 17$. Resuelva usando bisección y secante con $[17, 22.2]$

4.70 Consideremos la ecuación $x^5/5 + x^4/4 = 0$. Encuentre x_0 y x_1 de tal manera que el cero $x = -1.25$ este entre ambos pero que el método de la secante aproxime el
otro cero $x = 0$.

4.71 Considere $f(x) = x - \cos\left(\dfrac{0.785 - x\sqrt{1+x^2}}{1+2x^2}\right)$. Como se observa en la figura (**??**), esta función tiene un *cero* cerca de $x = 1$. Use el método de la secante para aproximar este cero con error absoluto estimado menor que 0.5×10^{-5}

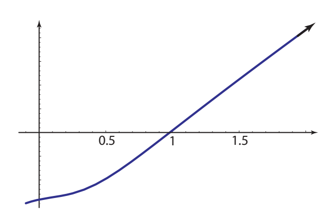


Figura 4.26

4.72 El método de la secante no requiere que el cero este entre x_0 y x_1. En el ejercicio anterior, aplique el método de la secante con $x_0 = 0$ y $x_1 = 0.1$

4.73 Resuelva la ecuación $\ln^2 x - x - 1 = 0$.

4.74 Considere la función $f(x) = \text{sgn}(x-1)\sqrt{|x-1|}$ (figura 4.74) donde $\text{sgn}(u) = \begin{cases} 1 & \text{si } u > 0 \\ 0 & \text{si } u = 0 \\ -1 & \text{si } u < 0 \end{cases}$. Use el método de la secante para aproximar el cero $x^* = 1$.

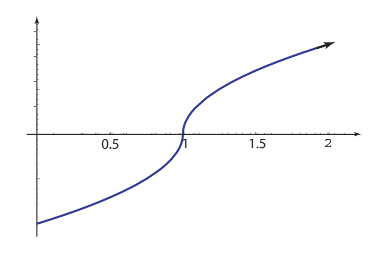


Figura 4.27

La fórmula para el método de la secante se podría modificar un poco más para evitar problemas de sobreflujo o división por cero.

Si $|f(x_{n-1})| \geq |f(x_n)| > 0$ entonces reescribimos la fórmula como

$$x_{k+1} = x_k + \frac{s_k}{1-s_k}(x_k - x_{k-1}), \;\; s_k = \frac{f(x_k)}{f(x_{k-1})}$$

donde la división por $1 - s_k$ se hace solo si $1 - s_k$ es "suficientemente" grande. Implemente esta versión del método y aplíquelo a los ejercicios anteriores.

4.18 Secante: Teorema de Convergencia. Orden de Convergencia.

El esquema iterativo

$$x_{k+1} = x_k - f(x_k) \cdot \frac{(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}$$

lo podemos reescribir como

$$x_{k+1} = x_k - c_k \;\; \text{con} \;\; c_k = f(x_k) \cdot \frac{(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}$$

así, c_k es una corrección de x_k. Esta corrección está compuesta por $f(x_k)$ y la pendiente de la secante $\dfrac{f(x_k) - f(x_{k-1})}{(x_k - x_{k-1})}$. Hablando grosso modo, si la diferencia $x_k - x_{k-1}$ se hace pequeña lentamente, la corrección más bien esta fuertemente influenciada por el valor del factor $f(x_k)$. Entonces lo que está localizando el método de la secante es un punto dónde f es "pequeña".

En este método no se puede asegurar que cada intervalo $[x_{n-1}, x_n]$ contenga al menos un cero (como bisección). Lo que si se puede asegurar es que converge localmente, es decir converge si las aproximaciones iniciales están suficientemente cerca de la raíz y además que la convergencia es superlineal con orden al menos de $(1+\sqrt{5})/2 = 1.61803$ para ceros simples. ¿Qué significa que "las aproximaciones estén suficientemente cercanas a un cero"?. La situación es similar a la del método de Newton. Para 'amarrar" el resultado debemos restringirnos a un intervalo "suficientemente pequeño" de tal manera que se pueda asegurar que se cumplen algunas condiciones que nos permiten concluir con la convergencia.

La idea del teorema de convergencia es esta: Si x^* es un cero simple de f, usando el teorema de Taylor en $I_\varepsilon = \{x \in \mathbb{R} : |x - x^*| \leq \varepsilon\}$, establecemos que

$$f(x) = (x - x^*)f'(x^*)\left(1 + \frac{x - x^*}{2}\frac{f''(\xi)}{f'(x^*)}\right), \;\; \xi \in I_\varepsilon.$$

x^* sería el único cero en I_ε si los tres factores de la derecha son distintos de cero, y para esto es suficiente pedir que I_ε es lo suficientemente pequeño como para que

$$\frac{x - x^*}{2}\frac{f''(\xi)}{f'(x^*)} \leq \varepsilon \max_{s,t \in I_\varepsilon}\left|\frac{f''(s)}{2f'(t)}\right| < 1$$

además, si ponemos $M_\varepsilon = \max\limits_{s,t \in I_\varepsilon}\left|\dfrac{f''(s)}{2f'(t)}\right|$,

$$|x_{n+1} - x^*| \leq \varepsilon^2 \left| \frac{f''(\xi_2)}{2f'(\xi_1)} \right| \leq \varepsilon \cdot \varepsilon M_\varepsilon < \varepsilon$$

lo que aplicado repetidamente nos lleva a

$$|x_n - x^*| \leq (\varepsilon M_\varepsilon)^{n-1} |x_1 - x^*|$$

Con lo cual establecemos la convergencia.

La frase x_0, x_1 son "aproximaciones estén suficientemente cercanas" a un cero x^* significan $x_0, x_1 \in I_\varepsilon$ con ε suficientemente pequeño de tal manera que

$$\varepsilon \max_{s,t \in I_\varepsilon} \left| \frac{f''(s)}{2f'(t)} \right| < 1$$

Para establecer los resultados necesitamos un desglose previo. Restamos x^* a ambos lados de $x_{n+1} = x_n - f(x_n) \cdot \dfrac{(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}$ y usamos diferencia divididas

$$\begin{aligned}
x_{n+1} - x^* &= x_n - x^* - f(x_n) \cdot \frac{(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})} \\
&= (x_n - x^*) \left(1 - \frac{f(x_n) - f(x^*)}{(x_n - x^*) f[x_{n-1}, x_n]} \right) \\
&= (x_n - x^*) \left(1 - \frac{f[x_n, x^*]}{f[x_{n-1}, x_n]} \right) \\
&= (x_n - x^*) \left(\frac{f[x_{n-1}, x_n] - f[x_n, x^*]}{f[x_{n-1}, x_n]} \right)
\end{aligned}$$

Ahora, como $f[x_{n-1}, x_n, x^*] = \dfrac{f[x_n, x^*] - f[x_{n-1}, x_n]}{x_{n-1} - x^*}$, entonces

$$(x_{n+1} - x^*) = (x_n - x^*)(x_{n-1} - x^*) \frac{f[x_{n-1}, x_n, x^*]}{f[x_{n-1} - x_n]}, \ n = 1, 2, ... \tag{4.13}$$

Esto lo vamos a usar en el siguiente teorema.

Teorema 4.15

Sea x^* un cero simple de f. Sea $I_\varepsilon = \{x \in \mathbb{R} : |x - x^*| \leq \varepsilon\}$. Supongamos que $f \in C^2[I_\varepsilon]$. Para un ε suficientemente pequeño se define

$$M_\varepsilon = \max_{s,t \in I_\varepsilon} \left| \frac{f''(s)}{2f'(t)} \right| \tag{4.14}$$

y asumamos que ε es lo suficientemente pequeño de tal manera que se cumpla $\varepsilon M_\varepsilon < 1$. Entonces, el método de la secante converge a una *única* raíz $x^* \in I_\varepsilon$ para cualesquiera dos aproximaciones iniciales $x_0 \neq x_1$ en I_ε.

Observe que el teorema exige que ε sea *suficientemente pequeño* de tal manera que $\left| \dfrac{f''(s)}{2f'(t)} \right|$ permanezca acotada en I_ε y que $\varepsilon M_\varepsilon < 1$.

Prueba. La suposición $\varepsilon M_\varepsilon < 1$ se puede hacer ya que M_ε es decreciente y

$$\lim_{\varepsilon \longrightarrow 0} M_\varepsilon = \frac{f''(x^*)}{2f'(x^*)} < \infty$$

Lo primero que hay que notar es que x^* es el único cero de f en I_ε. En efecto, según la fórmula de Taylor tendríamos

$$f(x) = f(x^*) + (x - x^*)f'(x^*) + \frac{(x - x^*)^2}{2} f''(\xi), \quad \text{con } \xi \text{ entre } x \text{ y } x^*,$$

Luego si $x \in I_\varepsilon$ también $\xi \in I_\varepsilon$ y tendríamos

$$f(x) = (x - x^*)f'(x^*) \left(1 + \frac{x - x^*}{2} \frac{f''(\xi)}{f'(x^*)} \right), \quad \xi \in I_\varepsilon. \tag{4.15}$$

Si $x \neq x^*$, los tres factores a la derecha de (4.15), son distintos de cero. En particular, el último es distinto de cero pues por hipótesis

$$\left| \frac{x - x^*}{2} \frac{f''(\xi)}{f'(x^*)} \right| \leq \varepsilon M_\varepsilon < 1$$

Por tanto, f solo se anula en $x = x^*$ en I_ε.

Ahora vamos a probar la convergencia. La idea es mostrar que todos los x_n's están en I_ε y que las aproximaciones consecutivas son distintas (excepto que para algún n suceda que $f(x_n) = 0$, en cuyo caso $x_n = x^*$ y hay convergencia en un número finito de pasos).

La prueba es por inducción. Supongamos que $x_n, x_{n-1} \in I_\varepsilon$ para algún n y que $x_n \neq x_{n-1}$. Por hipótesis esto es cierto para $n = 1$. Ahora, como $f \in C^2[I_\varepsilon]$

$$\begin{array}{rcl} f[x_{n-1},x_n] & = & f'(\xi_1) \\ & & \qquad\qquad \xi_i \in I_\varepsilon,\ i=1,2 \ \text{(ver ejercicio 3.6)}. \\ f[x_{n-1},x_n,x^*] & = & \frac{1}{2}f''(\xi_2) \end{array}$$

entonces, usando (4.13),

$$|x_{n+1}-x^*| \leq \varepsilon^2 \left|\frac{f''(\xi_2)}{2f'(\xi_1)}\right| \leq \varepsilon\cdot\varepsilon M_\varepsilon < \varepsilon$$

de donde $x_{n+1} \in I_\varepsilon$. Además, por (**??**) $x_{n+1} \neq x_n$ excepto que $f(x_n) = 0$ y en este caso $x_n = x^*$.

Por último, de (4.13) tenemos

$$|x_{n+1}-x^*| \leq |x_n - x^*|\,\varepsilon M_\varepsilon,\ \ n = 1,2,...,$$

lo que aplicado repetidamente nos lleva a

$$|x_n - x^*| \leq (\varepsilon M_\varepsilon)^{n-1}\,|x_1 - x^*|$$

Y como $\varepsilon M_\varepsilon < 1$ se sigue que $x_n \longrightarrow x^*$ si $n \longrightarrow \infty$.

El teorema es de evidente valor teórico pues construir I_ε requiere conocer x^*. Aún así vamos a dar un ejemplo sencillo.

Ejemplo 4.29

Considere $f(x) = x^2 - 1$. Vamos a encontrar un intervalo I_ε suficientemente pequeño, que contenga la raíz simple $x^* = 1$, en el que el método de la secante converge para cualesquiera x_0, x_1 distintos en I_ε. Como la función es sencilla, esto se puede hacer por inspección. Observe que en este caso

$$M_\varepsilon = \text{máx}_{t\in I_\varepsilon}\left|\frac{2}{4t}\right|$$

así que si $\varepsilon = 1/10$, $M_\varepsilon = 5/11$ (observe que I_ε es cerrado) y además $\varepsilon M_\varepsilon < 1$.

Entonces, en resumen, todas las hipótesis del teorema se cumplen. Por lo tanto el método de la secante se garantiza que converge a la raíz $x^* = 1$ para cualesquiera x_0, x_1 distintos en $I_{1/10} = [0.9, 1.1]$.

EJERCICIOS

4.75 En el ejemplo (4.29) muestre que basta con tomar cualquier $\varepsilon \in]0,2/3[$.

4.76 Considere $f(x) = x^2 - A$ con $A > 0$.

a) Explique porqué, para poder aplicar el teorema (4.15), solo podemos considerar ε tal que $\sqrt{A} - \varepsilon > 0$.

b) Determine los valores de ε (en términos de $\sqrt{A}$) de tal manera que, a la luz del teorema (4.15), podamos asegurar que el método de la secante converge a la *única* raíz $\sqrt{A} \in I_\varepsilon$ para cualesquiera dos aproximaciones iniciales $x_0 \neq x_1$ en I_ε.

4.77 Considere $f(x) = x^3 - 1$. Determine ε de tal manera que, a la luz del teorema (4.15), podamos asegurar que el método de la secante converge a la *única* raíz $1 \in I_\varepsilon$ para cualesquiera dos aproximaciones iniciales $x_0 \neq x_1$ en I_ε.

4.19 Secante: Orden de convergencia.

Ahora nos ocuparemos de la demostración formal del teorema acerca del orden de convergencia. La idea es esta: poniendo $e_n = x_n - x^*$ en (4.13) tenemos

$$e_{n+1} = e_n e_{n-1} \frac{f[x_{n-1}, x_n, x^*]}{f[x_{n-1} - x_n]}, \quad n = 1, 2, ... \tag{4.16}$$

De aquí se sigue que si x^* es un cero simple ($f(x^*) = 0$, $f'(x^*) \neq 0$), $x_n \longrightarrow x^*$ y si f'' existe y es continua en las *cercanías* de x^*, entonces $f[x_{n-1}, x_n, x^*] \longrightarrow \frac{1}{2} f''(x^*)$ y $f[x_{n-1} - x_n] \longrightarrow f'(x^*)$, así

$$\lim_{n \longrightarrow \infty} \frac{x_{n+1} - x^*}{x_n - x^*} = 0$$

es decir, la convergencia debe ser más que lineal: $x_n - x^*$ domina a $x_{n+1} - x^*$, pero $(x_n - x^*)^q$ si podría ser proporcional a $x_{n+1} - x^*$.

Para establecer el orden de convergencia de manera intuitiva, usamos un razonamiento informal que nos lleva a la sucesión de Fibonacci. Reescribimos (4.16) como

$$|e_{n+1}| = |e_n e_{n-1}| C, \quad \text{con} \quad C > 0$$

ahora multiplicamos a ambos lados por C y ponemos $E_n = C|e_n|$ y entonces

$$E_{n+1} = E_n E_{n-1}, \quad E_n \longrightarrow 0 \;\; \text{si} \;\; n \longrightarrow \infty$$

Ahora tomamos logaritmos a ambos lados y definimos $y_n = \ln(1/E_n)$, y queda la conocida sucesión de Fibonacci

$$y_{n+1} = y_n + y_{n-1} \tag{4.17}$$

La sucesión se resuelve para y_n resolviendo la ecuación característica $t^2 - t - 1 = 0$. Esta ecuación tiene dos soluciones $t_1 = (1+\sqrt{5})/2$ y $t_2 = (1-\sqrt{5})/2$. La solución general de (4.17) es

$$y_n = c_1 t_1^n + c_2 t_2^n, \quad \text{con} \quad c_1, c_2 \quad \text{constantes.}$$

Si $n \longrightarrow \infty$ entonces $y_n \longrightarrow \infty$ y debe ser $c_1 \neq 0$ (pues $t_2^n \longrightarrow 0$). Luego si $n \longrightarrow \infty$ entonces

$$y_n \sim c_1 t_1^n$$

de donde, pasando y_n a términos de e_n y poniendo $q = t_1$

$$\begin{aligned} \frac{1}{e_n} &\sim Ce^{c_1 q^n} \\ \frac{e_{n+1}}{e_n^q} &\sim \frac{C^q e^{c_1 q^n \cdot q}}{C e^{c_1 q^{n+1}}} = C^{q-1} \end{aligned}$$

por lo que $t_1 = 1.61803...$ sería el orden de convergencia. Veamos ahora todo lo anterior de manera rigurosa.

Teorema 4.16

Asumiendo las hipótesis del teorema anterior, el orden de convergencia del método de la secante es como mínimo $p = (1+\sqrt{5})/2 = 1.61803...$

Prueba. Supongamos que $x_0, x_1 \in I_\varepsilon$ y que todos los x_n's son distintos. Entonces $x_n \neq x^*$ y $x_n \longrightarrow x^*$ conforme $n \longrightarrow \infty$.

El número p del teorema satisface la ecuación

$$p^2 = p + 1 \tag{4.18}$$

De (4.13) se sigue

$$|x_{n+1} - x^*| \leq |x_n - x^*|\,|x_{n-1} - x^*| \cdot M_\varepsilon. \tag{4.19}$$

Sea

$$E_n = M_\varepsilon |x_n - x^*| \tag{4.20}$$

Entonces, multiplicando (4.19) por M_ε tendremos

$$E_{n+1} = E_n E_{n-1}$$

Luego

$$E_n \leq E^{p^n}, \quad E = \text{máx}\left(E_0, E_1^{1/p}\right) \tag{4.21}$$

En efecto, procedemos por inducción. El resultado es cierto para $n = 0$ y $n = 1$. Supongamos ahora que (4.21) es cierto para n y $n-1$. Entonces, usando (4.18)

$$E_{n+1} \leq E_n E_{n-1} \leq E^{p^n} E^{p^{n-1}} = E^{p^{n-1}(p+1)} = E^{p^{n-1}p^2} = E^{p^{n+1}},$$

Ahora, de (4.20)

$$|x_n - x^*| \leq \varepsilon_n, \;\; \varepsilon_n = \frac{1}{M_\varepsilon} E^{p^n}$$

Puesto que $E_0 = M|x_0 - x^*| \leq \varepsilon M_\varepsilon < 1$ y lo mismo pasa para E_1 entonces $E < 1$. Ahora basta notar que

$$\frac{\varepsilon_{n+1}}{\varepsilon_n^p} = M_\varepsilon^{p-1} \frac{E^{p^{n+1}}}{E^{p^n \cdot p}} = M_\varepsilon^{p-1}, \quad \text{para toda} \quad n.$$

Newton vs Secante.

Como $x_{n+1} = x_n - f(x_n) \cdot \dfrac{(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}$, cada paso en el método de la secante requiere solo una evaluación adicional de f, o sea que dos iteraciones en el método de la secante son a lo sumo 'tan costosas' como una iteración en el método de Newton. Ahora, puesto que $E^{p^{i+2}} = \left(E^{p^i}\right)^{p^2} = \left(E^{p^i}\right)^{p+1}$, dos iteraciones en el método de la secante llevan a un método de orden $p^2 = p + 1 = 2.618...$ Así que con un esfuerzo parecido, el método de la secante converge *localmente* más rápido que el método de Newton.

Al igual que el método de Newton, el método de la secante exhibe convergencia *lineal* para el caso de raíces múltiples (ver por ejemplo [19])

4.20 Un Método Híbrido: Secante-Bisección

Uno de los problemas del método de la secante es que aunque un cero x^* de f esté entre las aproximaciones iniciales x_0 y x_1, no se garantiza que el método converja (si converge) a x^*, además de posibles "caídas".

Ejemplo 4.30

Consideremos $f(x) = x^{20} - 1$. Si aplicamos el método de la secante con $x_0 = 0.5$ y $x_1 = 1.5$ tenemos un comportamiento no deseable, tal y como se muestra en la tabla (4.4)

n	x_n	Error estimado
2	0.5003007284	0.9996992716
3	0.5006013663	0.0003006379
4	25769.46097	25768.96037
5	0.5006013663041813	25768.96037
6	0.5006013663041813	0

Tabla 4.4 Problemas con el método de la secante

Para evitar comportamientos no deseables y asegurar que siempre la nueva iteración esté en un intervalo que contenga al cero que se quiere aproximar, combinamos el método de la secante con el método de bisección.

Para describir el procedimiento usamos $int\{b,c\}$ para denotar el más pequeño intervalo cerrado que contiene a b y c, es decir $int\{b,c\}$ es $[c,b]$ o $[b,c]$.

El algoritmo es como sigue: iniciamos con un intervalo $[a,b]$ en el que $f(a)$ y $f(b)$ tengan signos opuestos. Inicialmente el intervalo de bisección es $[c,b]$ pues iniciamos con $c = a$. En el proceso se actualizan a, b y c de la siguiente manera:

i. Si $f(a)$ y $f(b)$ tienen signos opuestos, se aplica una iteración del método de la secante (la cual actualiza a y b)

ii. Si $f(a)$ y $f(b)$ tienen signos iguales, se aplicará una iteración del método de la secante solo si se conoce que x_{n+1} estará en $int\{c,b\}$. Sino, se aplica bisección tomando como b el punto medio del intervalo $int\{c,b\}$.

iii. Finalmente se actualiza el intervalo $int\{c,b\}$ actualizando c por comparación del signo de $f(c)$ y $f(b)$.

En la figura (4.28) se muestra un caso particular con las primeras dos iteraciones.

La figura (4.29) ilustra una posible actualización de c.

El algoritmo requiere saber a priori si $x_{n+1} \in int\{c,b\}$ pero sin aplicar la iteración (para evitar una posible "caída" evitamos la división por $f(x_n) - f(x_{n-1})$). Digamos que $int\{c,b\} = [p,q]$, entonces

$$x_{n+1} = x_n - f(x_n) \cdot \frac{(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})} \in [p,q]$$

si y sólo si

$$p \leq x_n - f(x_n) \cdot \frac{(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})} \leq q$$

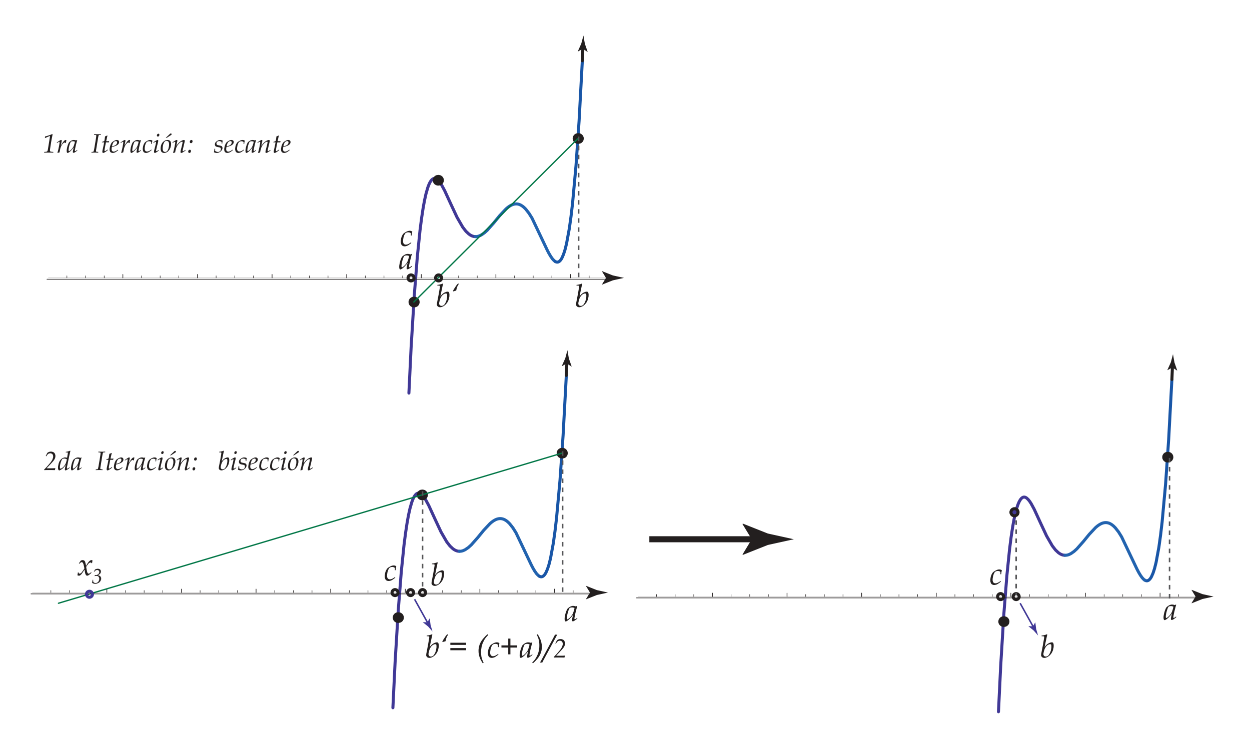


Figura 4.28

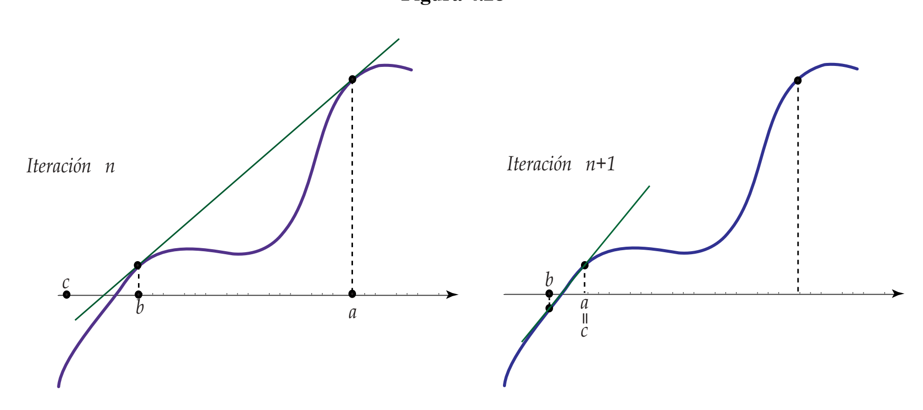


Figura 4.29

si ponemos $\Delta f = f(x_n) - f(x_{n-1})$ entonces $x_{n+1} \in [p,q]$ si y sólo si se cumple alguna de las dos condiciones siguientes

i.) si $\Delta f > 0$ entonces $(p - x_n)\Delta f < -f(x_n)(x_n - x_{n-1}) < (q - x_n)\Delta f$

ii.) si $\Delta f < 0$ entonces $(p - x_n)\Delta f > -f(x_n)(x_n - x_{n-1}) > (q - x_n)\Delta f$

4.21 Híbrido: Algoritmo e Implementación.

En este algoritmo, `TestSecante` hace la prueba $x_{n+1} \in int\{c,b\}$, es decir, determina si la siguiente iteración de secante estará en le intervalo $int\{c,b\}$.
Si es el caso, se aplica el método de la secante con $x_0 = a$ y $x_1 = b$ y se procede a la actualización de a y b, es decir

$a = x_1$ y $b = x_2$. Si los argumentos se reciben por referencia entonces son modificados en la subrutina. En código VBA Excel sería

```
Sub secanteItr(x0, x1)
If f(x1) <> f(x0) Then
 x2 = x1 - f(x1) * ((x1 - x0) / (f(x1) - f(x0)))
 x0 = x1
 x1 = x2
 End If

End Sub
```

Algoritmo 4.6: Híbrido Secante-Bisección.

Datos: una función continua f y $a,\ b$ tal que $f(a)f(b) < 0$ y δ
Salida: una aproximación del cero.

```
c = a;
n = 0;
repeat
    if f(a)f(b) < 0 then
        Aplicar SecanteItr(a,b)
    else
        if TestSecante =true then
            Aplicar SecanteItr(a,b)
        else
            Aplicar bisección b = b − 0.5(c − b);
    Intervalo para bisección;
    if f(a)f(b) < 0 then
        c = a
    n = n + 1
until (|c − b| < δ(|b| + 1)) Or (n > maxItr Or f(b) < δ);
return b;
```

Implementación en VBA Excel. Para hacer una hoja Excel, usamos como referencia la figura (4.30).

	A	B	C	D	E	F	G	H	I
1									
2			Híbrido Secante Bisección						
3									
4	f(x) =	x ^ 5 - 100x ^ 4 + 3995 x ^ 3 - 79700x ^ 2 + 794004x - 3160080							
5									
6							Método	Error	
7	a	b	delta	MaxItr	b	c	usado	Estimado	f(b)
8	21.34	22.45	5E-11	20	22.450000	21.340000	Secante1	0.097000404	-2.961297207
9					21.437000	22.450000	Secante1	0.903932827	-3.460053664
10					21.546067	22.450000	Bisección	0.451966414	-0.047000848

Figura 4.30 Hoja Excel para el híbrido secante-biseccion.

La subrutina principal es `HSecanteBiseccion`. Esta subrutina usa el evaluador de funciones clsMathParser para leer y evaluar la función f. También implementamos dos funciones `secanteItr(a,b,f)` y `x2estaEntre(c,b,a,f)`. Ambas son llamadas desde `HSecanteBiseccion` y usan la función f. `secanteItr(a,b,f)` hace una iteración del método de la secante y actualiza a y b.

 **Software:** Cuadernos LibreOffice y Excel

Código VBA 4.13: Híbrido Secante-Bisección

```
Option Explicit
Private Sub CommandButton1_Click()
Dim fx, a, b, delta, maxItr
    fx = Cells(4, 2)
    a = Cells(8, 1)
    b = Cells(8, 2)
    delta = Cells(8, 3)
    maxItr = Cells(8, 4)
    Call HSecanteBiseccion(fx, a, b, delta, maxItr, 8, 5)
End Sub

Sub HSecanteBiseccion(fx, aa, bb, delta, maxItr, fi, co)
Dim f As New clsMathParser
Dim okf As Boolean
Dim test1 As Boolean
Dim test2 As Boolean
Dim a, b, n, c, fa, fb

okf = f.StoreExpression(fx)
If Not okf Then
    MsgBox ("error en f: " + f.ErrorDescription)
    Exit Sub
End If

a = aa
b = bb
c = a
fa = f.Eval1(a)
fb = f.Eval1(b)
n = 0
Do
    If Sgn(fa) <> Sgn(fb) Then
       Cells(fi + n, co) = b
       Cells(fi + n, co + 1) = c
       Call secanteItr(a, b, f) 'Actualiza a y b
       Cells(fi + n, co + 2) = "Secante1"
       Cells(fi + n, co + 3) = Abs(b - c)
    Else
       If x2estaEntre(b, c, a, f) Then
          Cells(fi + n, co) = b
```

```
          Cells(fi + n, co + 1) = c
          Call secanteItr(a, b, f) 'Modifica a y b
          Cells(fi + n, co + 2) = "Secante2"
          Cells(fi + n, co + 3) = Abs(b - c)
        Else
        'bisección
          Cells(fi + n, co) = b
          Cells(fi + n, co + 1) = c
          b = b + 0.5 * (c - b)
          Cells(fi + n, co + 2) = "Bisecci\'on"
          Cells(fi + n, co + 3) = Abs(b - c)
       End If
    End If
    fa = f.Eval1(a)
    fb = f.Eval1(b)
    Cells(fi + n, co + 4) = fb
    If Sgn(fa) = Sgn(fb) Then
       'nada
    Else: c = a
    End If
n = n + 1
Loop Until Abs(c - b) < delta Or n >= maxItr Or Abs(fb) < delta
End Sub

Sub secanteItr(x0, x1, f) 'Modifica x0 y x1
Dim x2, fx1, fx0
 fx1 = f.Eval1(x1)
 fx0 = f.Eval1(x0)
 x2 = x1 - fx1 * ((x1 - x0) / (fx1 - fx0))
 x0 = x1
 x1 = x2
End Sub

Function x2estaEntre(b, c, a, f)
Dim test1 As Boolean
Dim test2 As Boolean
Dim q, x0, x1, p, pf, Df, x2, fx1, fx0

q = Application.Max(b, c)
p = Application.Min(b, c)
x0 = a
x1 = b
fx1 = f.Eval1(x1)
fx0 = f.Eval1(x0)
Df = fx1 - fx0
pf = AAAAAAAA * (x1 - x0)
test1 = (Df > 0 And (p - x1) * Df < pf And pf < (q - x1) * Df)
test2 = (Df < 0 And (p - x1) * Df > pf And pf > (q - x1) * Df)
```

```
x2estaEntre = test1 Or test2
End Function
```

Ejemplo 4.31

Considere la función $f(x) = x^5 - 100x^4 + 3995x^3 - 79700x^2 + 794004x - 3160080$. Sus ceros son $x^* = 18, 19, 20, 21, 22$. En la tabla (**??**) se muestra los valores de b y c al aplicar el método híbrido secante-bisección con $x_0 = 21.34$ y $x_1 = 22.45$ (para aproximar $x^* = 22$)

n	b	c	Método
1	21.4370004	21.34	Secante1
2	21.54606717	22.45	Secante1
3	21.99803359	22.45	Bisección
4	22.00708175	22.45	Secante2
5	21.99997119	21.99803359	Secante1
6	21.99999958	22.00708175	Secante1
7	22	22.00708175	Secante2

En la línea 3 se observa que $b = 21.54606717$ y $c = 22.45$. En la siguiente iteración se usa bisección. Esto sucede porque si se aplica una iteración de la secante con los valores "actuales" $a = 21.4370004035587, b = 21.5460671728629$ (para los cuales no hay cambio de signo) entonces el valor futuro habría sido $x_{n+1} = 20.7894316058807$ y este valor *no* está en el intervalo actual $[c, b] = [22.45, 21.54606717]$, Entonces se usó bisección. Si no se hubiera hecho esto, entonces el método de la secante (clásico) hubiera convergido al cero $x^* = 21$ que no es el que está en el intervalo inicial.

Ejemplo 4.32

En la tabla se muestra los valores de b y c al aplicar el método híbrido secante-bisección, con $x_0 = 0.5$ y $x_1 = 2$, para aproximar el cero $x^* = 1$ de $f(x) = x^{20} - 1$.
Recordemos es desempeño desastroso del método de la secante, que se mostró en la tabla (4.4), para este mismo problema.

n	b	c	Método	$\|b-c\|$
1	2.000000	0.500000	Secante1	1.43051×10^{-6}
2	0.500001	2.000000	Secante1	1.499997139
3	0.500003	2.000000	Bisección	0.749998569
4	1.250001	0.500001	Secante1	0.008646705
5	0.508648	1.250001	Secante1	0.73280628
6	0.517195	1.250001	Bisección	0.36640314
7	0.883598	1.250001	Bisección	0.18320157
8	1.066800	0.508648	Secante1	0.153141481
9	0.661790	1.066800	Secante1	0.293907403
1	0 0.772892	1.066800	Bisección	0.146953701
11	0.919846	1.066800	Bisección	0.073476851
12	0.993323	1.066800	Secante2	0.025927918
13	1.040871942	0.99332301	Secante1	0.004405128
14	0.997728138	1.040871942	Secante1	0.041636217
15	0.999235725	1.040871942	Secante2	0.040855271
16	1.000016672	0.999235725	Secante1	0.000764154
17	0.999999879	1.000016672	Secante1	1.66716×10^{-5}
18	1	1.000016672	Secante2	1.66716×10^{-5}

EJERCICIOS

4.78 Considere la función $f(x) = x^5 - 100x^4 + 3995x^3 - 79700x^2 + 794004x - 3160080$. Aplique el método de la secante con $x_0 = 21.34$ y $x_1 = 22.45$ y compare con el ejemplo ().

4.79 Aplique el método de la secante clásico a la ecuación $x^{20} - 1 = 0$.

4.80 Uno de los pasos delicados del algoritmo es la instrucción

```
If Sgn(f(a)) = Sgn(f(b)) Then
      'nada
   Else: c = a
```

Explique porqué se actualiza c de esta manera.

4.22 Interpolación Inversa.

Supongamos que f es una función monótona en un intervalo I alrededor de uno de sus ceros x^*. Si tenemos dos aproximaciones x_0 y x_1 de este cero en I entonces, si $x_0 = f^{-1}(y_0)$ y $x_1 = f^{-1}(y_1)$, podemos construir la siguiente tabla de diferencias divididas

y	x	
y_0	x_0	
y_1	x_1	$f^{-1}[y_0, y_1]$

Como queremos aproximar $x^* = f^{-1}(0)$ entonces

- la aproximación x_2 la podemos obtener por interpolación lineal

$$x_2 = x_0 + (0 - y_0) f^{-1}[y_0, y_1]$$

- la aproximación x_3 la podemos obtener por interpolación cuadrática.

  Si $y_2 = f(x_2)$ entonces (formalmente) $x_2 = f^{-1}(y_2)$ y actualizamos la tabla

y	x		
y_0	x_0		
y_1	x_1	$f^{-1}[y_0, y_1]$	
y_2	x_2	$f^{-1}[y_1, y_2]$	$f^{-1}[y_0, y_1, y_2]$

  con lo que podemos calcular x_3

$$x_3 = x_2 + (0 - y_0)(0 - y_1) f^{-1}[y_0, y_1, y_2] = x_2 + y_0 y_1 f^{-1}[y_0, y_1, y_2]$$

y entonces $y_3 = f(x_3)$ y (formalmente) $x_3 = f^{-1}(y_3)$.

Suponiendo que y_0 y_1 son pequeños, entonces y_0y_1 es todavía más pequeño, haciendo que *la corrección* que se le esta sumando a x_2 sea pequeña.

Si es necesario, se puede actualizar de nuevo la tabla

y	x			
y_0	x_0			
y_1	x_1	$f^{-1}[y_0,y_1]$		
y_2	x_2	$f^{-1}[y_1,y_2]$	$f^{-1}[y_0,y_1,y_2]$	
y_3	x_3	$f^{-1}[y_2,y_3]$	$f^{-1}[y_1,y_2,y_3]$	$f^{-1}[y_0,y_1,y_2,y_3]$
$\ldots$	$\ldots$			

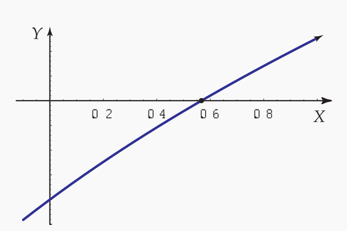
y calcular

$$x_4 = x_3 - y_0y_1y_2f^{-1}[y_0,y_1,y_2,y_3],\ \ y_4 = f(x_4),\ \ x_4 = f^{-1}(x_4)$$

En general, el proceso converge rápidamente: $x_k \longrightarrow x^*$ si $k \longrightarrow \infty$.

Ejemplo 4.33

Usemos este método para resolver $x - e^{-x} = 0$.



Necesitamos inicialmente dos pares (x_0,y_0) y (x_1,y_1). Si tomamos como aproximaciones iniciales $x_0 = 0.5$ y $x_1 = 0.6$ entonces $y_0 = -0.106531...$ y $y_1 = 0.0511884...$

- La primera tabla de diferencias divididas es

y	x	
-0.106531	0.5	
0.0511884	0.6	0.63403892

Por tanto

$$x_2 = 0.567545$$

Ejemplo 4.33 (continuación).

- La segunda tabla de diferencias divididas es

y	x		
-0.106531	0.5		
0.0511884	0.6	0.634039	
0.00062884	0.567545	0.641925	0.0735906

Por tanto $x_3 = 0.567143$

- La tercera tabla de diferencias divididas es

y	x			
-0.106531	0.5			
0.0511884	0.6	0.634039		
0.00062884	0.567545	0.641925	0.0735906	
-1.07985×10^{-6}	0.56714260	0.6381499	0.07374301	0.001430372

Por tanto $x_4 = 0.5671427538$

El análisis de convergencia de este algoritmo no es simple dada la complicada estructura de las derivadas sucesivas de las funciones inversas f^{-1}.

4.23 Interpolación Cuadrática Inversa.

Una variante muy usada de interpolación inversa es la llamada *Interpolación Cuadrática Inversa.*

Si estamos buscando ceros reales, no es conveniente interpolar con una parábola (en vez de una recta) para aproximar estos ceros. Como necesitamos la intersección con el eje x (la aproximación del cero), esto no sería posible si el discriminante es negativo.

En vez de una parábola $y = P(x)$ que interpola tres puntos (x_i, y_i), $i = 0,1,2$, buscamos la parábola $x = Q(y)$ que interpola los puntos (y_i, x_i), $i = 0,1,2$. Esta parábola siempre interseca al eje X en $x = Q(0)$.

Observe que este método requiere que $f(x_0), f(x_1)$ y $f(x_2)$ sean distintos. Por ejemplo, si tratamos de utilizar este método para calcular $\sqrt{a}$ con el polinomio $f(x) = x^2 - a$ y si comenzamos con $x_0 = -a, x_1 = 0$ y $x_2 = a$, entonces $f(x_0) = f(x_1)$ y no podríamos calcular el polinomio interpolante.

El método de interpolación cuadrática inversa es el siguiente

Ejemplo 4.34

Sea $P(x) = x^3 + x + 1$. La única solución real de la ecuación $P(x) = 0$ es $x^* = -0.6823278\ldots$. Usando los puntos

$$\{\,(-1, P(-1)),\,(-0.5, P(-0.5)),\,(0, P(0))\,\}$$

calculamos el polinomio interpolante con los datos

$$\{\,(P(-1), -1),\,(P(-0.5), -0.5),\,(P(0), 0)\,\}$$

El polinomio cuadrático (inverso) interpolante es

$$Q(y) = -0.718182 + 0.5y + 0.218182y^2$$

Y la aproximación a la solución es

$$x_3 = Q(0) = -0.718182$$

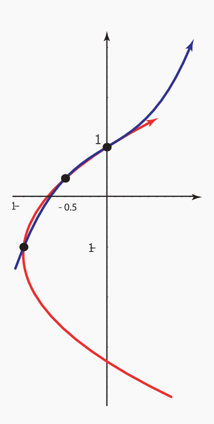


Figura 4.31 Cuadrática (inversa) interpolante.

- Si f es monótona en un intervalo adecuado, entonces si tenemos tres puntos distintos x_{n-2}, x_{n-1} y x_n, se puede calcular el polinomio interpolante $P_2(y)$ para los tres puntos $(y_{n-2}, x_{n-2}), (y_{n-1}, x_{n-1})$ y (y_n, x_n), donde $y_{n-i} = f(x_{n-i})$.
- $x_{n+1} = P_2(0)$
- Descartamos x_{n-2} y calculamos x_{n+2} de la misma manera como calculamos x_{n+1}, esta vez usando $(y_{n-1}, x_{n-1}), (y_n, x_n)$ y (y_{n+1}, x_{n+1}).

Formalmente,

Teorema 4.17

Supongamos que $f \in C^2[a,b]$ y que el cero $x^* \in [a,b]$. Si $f'(x^*) \neq 0$ entonces existe $\delta > 0$ tal que la iteración

$$x_{n+1} = x_{n-1} - f(x_{n-1})\,F_n, \quad n = 2, 3, \ldots$$

con

$$F_n = \frac{(f(x_{n-2}))^2(x_n - x_{n-1}) + f(x_{n-2})f(x_{n-1})(x_{n-1} - x_n) + (f(x_{n-1}) - f(x_n))f(x_n)(x_{n-2} - x_{n-1})}{(f(x_{n-1}) - f(x_{n-2}))(f(x_n) - f(x_{n-2}))(f(x_n) - f(x_{n-1}))},$$

converge a x^* para ciertas aproximaciones iniciales $x_0, x_1, x_2 \in [x^* - \delta,\, x^* + \delta]$.

El número de evaluaciones se puede reducir usando un truco algebraico

$$\begin{aligned}
R &= \frac{f(x_{n-1})}{f(x_n)} \\
S &= \frac{f(x_{n-1})}{f(x_{n-2})} \\
T &= \frac{f(x_{n-2})}{f(x_n)} \\
P &= S(T(R-T)(x_n - x_{n-1}) - (1-R)(x_{n-1} - x_{n-2})) \\
Q &= (T-1)(R-1)(S-1) \\
x_{n+1} &= x_{n-1} + \frac{P}{Q}, \quad k = 2,3,...
\end{aligned}$$

4.23.1 Algoritmo e Implementación en Excel.

Algoritmo 4.7: Interpolación Cuadrática Inversa.

Datos: una función f, las aproximaciones x_0, x_1, x_2, δ y maxItr

Salida: una aproximación p_3 a un cero de f.

$k = 2$;

$p_0 = x_0,\, p_1 = x_1,\, p_2 = x_2.$;

$p_3 = p_2,\, p_2 = p_1,\, p_1 = p_0.$;

while $|p_3 - p_2| < \delta \;\; o \;\; k < N$ **do**

  $p_0 = p_1,\, p_1 = p_2,\, p_2 = p_3.$;

  $f_0 = f(p_0),\, f_1 = f(p_1),\, f_2 = p(p_2)$;

  $R = \frac{f_1}{f_2},\; S = \frac{f_1}{f_0},\; T = \frac{f_0}{f_2},$;

  $P = S(T(R-T)(p_2 - p_1) - (1-R)(p_1 - p_0))$;

  $Q = (T-1)(R-1)(S-1)$;

  $p_3 = p_1 + \frac{P}{Q}$;

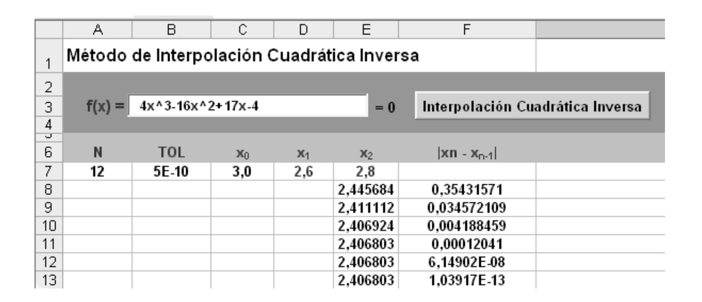
  $k = k + 1$;

end

return p_3;

Para la estimación del error se puede usar $|p_3 - p_2|$.

Implementación en Excel. La implementación que sigue corresponde a la hoja de la figura (**??**). Observe que en la implementación *no hay manejo de excepciones*. Para afinar el programa se deberá controlar los valores que retorna la función para verificar que no hay "infinitos", "Nan" ni números complejos.



N	TOL	x_0	x_1	x_2	$\|xn - x_{n-1}\|$
12	5E-10	3,0	2,6	2,8	
				2,445684	0,35431571
				2,411112	0,034572109
				2,406924	0,004188459
				2,406803	0,00012041
				2,406803	6,14902E-08
				2,406803	1,03917E-13

La implementación requiere importar la biblioteca clsMathParser (para leer y evaluar la función f) y agregar un campo de texto y un botón.

Código VBA 4.14: Interpolación Cuadrática Inversa con `clsMathParser`

```
Private Sub CommandButton1_Click()
   Call ICI
End Sub
Sub ICI()
 Dim Formula As String
 Dim FormulaEstaBien As Boolean
 Dim fun As New clsMathParser
 n = Cells(7, 1)
 tol = Cells(7, 2)
 p0 = Cells(7, 3)
 p1 = Cells(7, 4)
 p2 = Cells(7, 5)
      'leer fórmula
 Formula = TextBox1.Text
      'revisar sintaxis y asignarla como función actual
 FormulaEstaBien = fun.StoreExpression(Formula)
 If Not FormulaEstaBien Then
    MsgBox ("Error de sintaxis: " + fun.ErrorDescription)
    Exit Sub
 End If
   k = 2
   p3 = p2
   p2 = p1
   p1 = p0
   Do While k < n Or Abs(p3 - p2) < tol
      p0 = p1
      p1 = p2
      p2 = p3
      F0 = fun.Eval1(p0)
      F1 = fun.Eval1(p1)
      F2 = fun.Eval1(p2)
      r = F1 / F2
      s = F1 / F0
      t = F0 / F2
      p = s * (t * (r - t) * (p2 - p1) - (1 - r) * (p1 - p0))
```

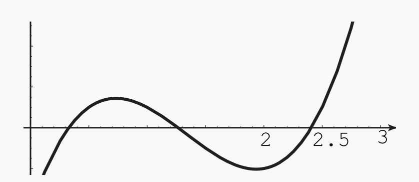
```
      Qu = (t - 1) * (r - 1) * (s - 1)
      p3 = p1 + p / Qu
      Cells(7 + k - 1, 5) = p3
      Cells(7 + k - 1, 6) = Abs(p3 - p2)
      k = k + 1
   Loop
End Sub
```

 **Software:** Cuadernos LibreOffice y Excel

Ejemplo 4.35

Aproxime la raíz más grande de $f(x) = 4x^3 - 16x^2 + 17x - 4$.

Solución: En la figura se puede ver la gráfica de la función f.



- Si iniciamos con los tres puntos $x_0 = 1.5$, $x_1 = 2$, $x_2 = 2.5$ el método aproxima la raíz $x^* = 1.26466...$

```
x0      x1              x2              |x_n - x_(n-1)|
-------------------------------------------------------
1,5     2               2,5
                        1,666666667     0.833333333
                        1,573593074     0.093073593
                        1,745140714     0.171547641
                        1,488651258     0.256489456
                        1,343629367     0.14502189
                        1,277823721     0.065805647
                        1,264876662     0.012947059
                        1,264658507     0.000218155
                        1,26465829      2,1718*10^(-7)
```

- Si iniciamos con los tres puntos $x_0 = 2$, $x_1 = 2.5$, $x_2 = 3$ el método aproxima la raíz correcta.

```
x0      x1              x2                  |x_n - x_(n-1)|
------------------------------------------------------------
2       2,5             3
                        2,3512820512821     0.648717949
                        2,4021859153538     0.050903864
                        2,4070625620152     0.004876647
                        2,4068034905016     0.000259072
                        2,4068032513232     2,39178*10^(-7)
                        2,4068032513242     9,40581*10^(-13)
```

El método de interpolación cuadrática inversa, como el de la secante, se debe usar con otro método porque, además de ser muy vulnerable a problemas asociados con la precisión de la máquina, no se garantiza que la nueva aproximación quede en un intervalo que contenga a la raíz x^*. Para aclarar esto supongamos que tenemos tres puntos $a < b < c$ y sean $A = f(a), B = f(b), C = f(c)$ y $d = p_2(0)$.

Si por ejemplo $AB < 0$ y $AC < 0$ entonces $x^* \in [a,b]$.

$$d = p_2(0) = \frac{BC}{(A-B)(A-C)} + \frac{AC}{(B-A)(B-C)} + \frac{AB}{(C-B)(C-A)}$$

Si $d \in [a,b]$ descartamos c y reordenamos los puntos de nuevo para calcular la siguiente aproximación. Pero podría pasar que $d \notin [a,b]$ o aún que d esté muy cercano de a o muy cercano a b. Entonces d no se puede usar. En vez de esto, usamos el intervalo $[a,b]$ con el método de la secante y si tenemos problemas, usamos bisección. El método de Brent se encarga de hacer una delicada y eficiente transición entre un método y otro, basado en el análisis de situaciones que pudieran provocar, entre otras cosas, *underflow, overflow* o división por cero.



5 INTEGRACIÓN NUMÉRICA.

5.1 Introducción

La integral definida $\int_a^b f(x)\,dx$ no siempre se puede calcular usando el teorema fundamental del cálculo porque hay funciones que no tienen primitiva elemental, es decir la integral indefinida no se puede expresar en términos de funciones elementales. En estos casos, las integrales definen una nueva función.

Ejemplo 5.1

a.) $\int e^{x^2}\,dx = \frac{\sqrt{\pi}}{2}\,\text{Erfi}(x)$

b.) $\int \frac{\operatorname{sen} x}{x}\,dx = \text{SinIntegral}(x)$

c.) $\int \frac{\cos x}{x}\,dx = \text{CosIntegral}(x)$

d.) $\int \frac{e^x}{x}\,dx = \text{ExpIntegralEi}(x)$

e.) Integral de Fresnel $C(z) = \int_0^z \cos(\pi x^2/2)\,dx$

f.) etc.

Aquí solo consideramos métodos de integración aproximada de la forma

$$\int_a^b f(x)\,dx = \sum_{i=1}^{n} w_i\,f(x_i) \tag{5.1}$$

donde los *nodos* $x_0 < x_1 < x_2 < ... < x_n$ están en $[a,b]$. A los w_i's se les llama "pesos".

5.2 Fórmulas de Newton-Cotes.

Las fórmulas de Newton-Cotes son fórmulas del tipo

$$\int_a^b w(x)f(x)\,dx = \sum_{i=0}^{n} w_i f(x_i) + E_n, \quad h=(b-a)/n, \quad x_i = a+i\cdot h.$$

Para determinar los pesos w_i se usa la fórmula de interpolación de Lagrange.

Sea $f\in C^{n+1}[a,b]$. Sea $P_n(x)$ el polinomio de grado $\leq n$ que interpola f en los $n+1$ puntos (distintos) $x_0,x_1,...,x_n$ en el intervalo $[a,b]$. Para cada valor fijo $x\in[a,b]$ existe $\xi(x)\in]a,b[$ tal que

$$f(x)-P_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x-x_0)(x-x_1)\cdots(x-x_n)$$

Entonces

$$\int_a^b f(x)\,dx = \int_a^b P_n(x) + \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x-x_0)(x-x_1)\cdots(x-x_n)\,dx \tag{5.2}$$

En particular, usando la forma de Lagrange del polinomio interpolante, $P_n(x) = y_0L_{n,0}(x)+y_1L_{n,1}(x)+...+y_nL_{n,n}(x)$ con $L_{n,k}(x) = \prod_{\substack{i=0\\ i\neq k}}^{n}\frac{x-x_i}{x_k-x_i}$, tenemos

Teorema 5.1

Sea $f\in C^{n+1}[a,b]$. Sea $P_n(x)$ el polinomio de grado $\leq n$ que interpola f en los $n+1$ puntos (distintos) $x_0,x_1,...,x_n$ en el intervalo $[a,b]$. Existe $\eta\in]a,b[$ tal que

$$\int_a^b f(x)\,dx = \sum_{k=0}^{n} y_k \int_a^b \prod_{\substack{i=0\\ i\neq k}}^{n}\frac{x-x_i}{x_k-x_i}\,dx + \frac{f^{(n+1)}(\eta)}{(n+1)!}\int_a^b \prod_{i=0}^{n}(x-x_i)\,dx. \tag{5.3}$$

siempre y cuando $\prod_{i=0}^{n}(x-x_i)$ sea de un mismo signo en $[a,b]$.

Nota: aquí usamos el teorema del valor medio para integrales: Si en $[a,b]$ G es continua y φ integrable y de un mismo signo, entonces existe $\eta\in]a,b[$ tal que $\int_a^b G(x)\varphi(x)\,dx = G(\eta)\int_a^b \varphi(x)\,dx$.

Puede pasar que $\int_a^b \prod_{i=0}^{n}(x-x_i)\,dx = 0$. En este caso se debe hacer un cambio en la fórmula del error. Para esto necesitamos el siguiente teorema:

Teorema 5.2

Sea $f \in C^{n+1}[a,b]$. Sean $x_0, x_1, ..., x_n$ $n+1$ puntos *no necesariamente distintos* en $[a,b]$, entonces

$$f(x) = \widehat{P}_n(x) + f[x_0, x_1, ..., x_n, x]\prod_{i=0}^{n}(x - x_i)$$

donde $\widehat{P}_n(x) = \sum_{i=0}^{n} f[x_0, x_1, ..., x_i]\prod_{i=0}^{i-1}(x - x_i)$ es un polinomio de grado $\leq n$ que interpola f en $x_0, x_1, ...x_n$ en el siguiente sentido: Si un punto u aparece repetido $k+1$ veces entre los números $x_0, x_1, ..., x_n$, entonces $\widehat{P}_n^{(j)}(x) = f^{(j)}(x)$, $m = 0, 1, ..., k$

Volvamos ahora a nuestro problema: Primero observemos que podemos reescribir la fórmula de error. Sea $E_n = \dfrac{f^{(n+1)}(\xi_x)}{(n+1)!}\prod_{i=0}^{n}(x - x_i)$.

$$E_n = f[x_0, x_1, ..., x_n, \xi_x]\prod_{i=0}^{n}(x - x_i)$$

Si x_{k+1} es un nodo *arbitrario,*

$$f[x_0, x_1, ..., x_k, x] = f[x_0, x_1, ..., x_k, x_{k+1}] + f[x_0, x_1, ..., x_{k+1}, x](x - x_{k+1})$$

Luego, si $\int_a^b \prod_{i=0}^{n}(x - x_i)\, dx = 0$, $f \in C^{n+2}[a,b]$ y si $\prod_{i=0}^{n+1}(x - x_i)$ mantiene el mismo signo en $[a,b]$, entonces

$$\begin{aligned}
\int_a^b E_n\, dx &= \int_a^b f[x_0, x_1, ..., x_k, x_{k+1}]\prod_{i=0}^{n}(x - x_i) + f[x_0, x_1, ..., x_{k+1}, x](x - x_{k+1})\prod_{i=0}^{n}(x - x_i)\, dx \\
&= \int_a^b f[x_0, x_1, ..., x_{k+1}, x]\prod_{i=0}^{n+1}(x - x_i)\, dx \\
&= \frac{f^{(n+2)}(\eta)}{(n+2)!}\int_a^b \prod_{i=0}^{n+1}(x - x_i)\, dx, \;\; \eta \in]a,b[
\end{aligned}$$

Resumiendo,

Teorema 5.3

Si $\int_a^b \prod_{i=0}^{n}(x - x_i)\, dx = 0$, $f \in C^{n+2}[a,b]$ y si $\prod_{i=0}^{n+1}(x - x_i)$ mantiene el mismo signo en $[a,b]$, entonces

$$\int_a^b f(x)\, dx = \sum_{k=0}^{n} y_k \int_a^b \prod_{\substack{i=0 \\ i\neq k}}^{n} \frac{x - x_i}{x_k - x_i}\, dx + \frac{f^{(n+2)}(\eta)}{(n+2)!}\int_a^b \prod_{i=0}^{n+1}(x - x_i)\, dx, \;\; \eta \in]a,b[\tag{5.4}$$

EJERCICIOS

5.1 Supongamos que f es integrable en el sentido de Riemann en $[a,b]$. Usando un solo punto de interpolación (x_0,y_0), $f(x) = y_0 + f[x_0,x](x-x_0)$. Usando el teorema (5.1) o el teorema (5.3), muestre que

a) Si $x_0 = a$, $\displaystyle\int_a^b f(x)\,dx = (b-a)f(a) + \frac{f'(\eta)(b-a)^2}{2}$

b) **(Regla del punto medio)** Si $x_0 = (a+b)/2$ y $x_0 = x_1$, entonces

$$\int_a^b f(x)\,dx = (b-a)f(x_0) + \frac{f''(\eta)(b-a)^3}{24}$$

5.2 Supongamos que f es integrable en el sentido de Riemann en $[a,b]$. Si partimos el intervalo en n subintervalos de tamaño $h = (b-a)/n$ y aplicamos a cada subintervalo $[x_i, x_{i+1}]$ la regla del punto medio, muestre que

$$\int_a^b f(x)\,dx = \frac{b-a}{n}\sum_{i=1}^{n} f(a+(i-1/2)h) + \frac{f''(\eta)(b-a)^3}{24n}, \;\; \eta \in]a,b[$$

5.3 Regla del Trapecio.

En la regla del trapecio, para aproximar $\displaystyle\int_a^b f(x)\,dx$ dividimos el intervalo $[a,b]$ en n subintervalos: si $h = (b-a)/n$ y $x_i = a + ih$, en cada subitervalo $[x_i,\, x_{i+1}]$, cambiamos la función f por el polinomio interpolante de grado 1 (figura 5.1).

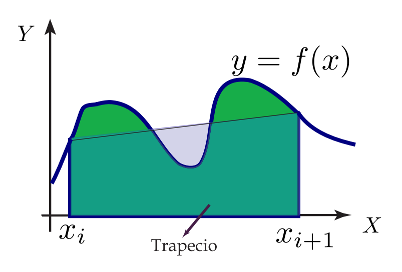


Figura 5.1 Regla del trapecio

$$\int_a^b f(x)\,dx = \sum_{i=0}^{n-1}\int_{x_i}^{x_{i+1}} f(x)\,dx$$

Para aproximar cada integral $\displaystyle\int_{x_i}^{x_{i+1}} f(x)\,dx$ necesitamos el polinomio que interpola a f en $(x_i, f(x_i))$, $(x_{i+1}, f(x_{i+1}))$:
$P(x) = f(x_i)\,\dfrac{(x-x_{i+1})}{h} + f(x_{i+1})\,\dfrac{(x-x_i)}{h} + E$ con $E = (x-x_i)(x-x_{i+1})\dfrac{f''(\xi(x))}{2}$ (si f'' es continua en $[x_i, x_{i+1}]$).

$$\begin{aligned}\int_{x_i}^{x_{i+1}} f(x)\,dx &= \int_{x_i}^{x_{i+1}} f(x_i)\,\frac{(x-x_{i+1})}{h} + f(x_{i+1})\,\frac{(x-x_i)}{h} + E\;dx\\ &= \frac{h}{2}[f(x_i)+f(x_{i+1})] + \int_{x_i}^{x_{i+1}} (x-x_i)(x-x_{i+1})\frac{f''(\xi(x))}{2}\;dx\end{aligned}$$

Para calcular $\int_{x_i}^{x_{i+1}} E\,dx$ necesitamos recordar el teorema del valor medio para integrales: Si en $[x_i, x_{i+1}]$ G es continua y φ integrable y de un mismo signo, entonces existe $\eta_i \in]x_i, x_{i+1}[$ tal que $\int_{x_i}^{x_{i+1}} G(x)\varphi(x)\,dx = G(\eta_i)\int_{x_i}^{x_{i+1}} \varphi(x)\,dx$.

Ahora, poniendo $G(x) = f''(\xi(x))/2$ y $\varphi(x) = (x-x_i)(x-x_{i+1})$, obtenemos

$$\int_{x_i}^{x_{i+1}} (x-x_i)(x-x_{i+1})\frac{f''(\xi_x)}{2}\;dx = -\frac{h^3}{12}f''(\eta_i),\;\; \eta_i \in]x_i, x_{i+1}[.$$

Observe que $(x-x_i)(x-x_{i+1})$ tiene el mismo signo sobre $[x_i, x_{i+1}]$ (siempre es negativa) y que $\int_{x_i}^{x_{i+1}} (x-x_i)(x-x_{i+1})\,dx = -h^3/6$.

Finalmente:

$$\int_{x_i}^{x_{i+1}} f(x)\,dx = \frac{h}{2}[f(x_i)+f(x_{i+1})] - \frac{h^3}{12}f''(\eta_i),\;\; \varepsilon_i \in [x_i, x_{i+1}]. \tag{5.5}$$

Si ponemos, para abreviar los cálculos, $G_i(x) = P_i(x) + (x-x_i)(x-x_{i+1})f''(\xi_i(x))/2$, con $P_i(x)$ el polinomio (lineal) interpolante en $[x_i, x_{i+1}]$, entonces

$$\begin{aligned}\int_a^b f(x)\,dx &= \int_{x_0}^{x_1} G_0(x)\,dx + \int_{x_1}^{x_2} G_1(x)\,dx + ... + \int_{x_{n-1}}^{x_n} G_{n-1}(x)\,dx\\ &= \frac{h}{2}\left(f(x_0)+f(x_n)+2\sum_{i=1}^{n-1} f(x_i)\right) - \frac{h^3}{12}\sum_{k=0}^{n-1} f''(\eta_k)\end{aligned}$$

donde $h = \dfrac{b-a}{n}$, $\eta_i \in [x_i,\, x_{i+1}]$ y $x_i = a + i\cdot h$, $i = 0,1,...,n$.

Podemos simplificar la fórmula observando que

$$-\frac{h^3}{12}\sum_{k=0}^{n-1} f''(\eta_k) = -\frac{h^2}{12}\cdot(b-a)\left[\frac{\sum_{k=0}^{n-1} f''(\eta_k)}{n}\right]$$

La expresión en paréntesis cuadrados es un *promedio* de los valores de f'' en $[a,b]$, por lo tanto este promedio está entre el máximo y el mínimo absoluto de f'' en $[a,b]$ (asumimos f'' continua). Finalmente, por el teorema del valor intermedio, existe $\xi \in]a,b[$ tal que $f''(\xi)$ es igual a este valor promedio, es decir

$$-\frac{h^3}{12}\sum_{k=0}^{n-1} f''(\eta_k) \;=\; -\frac{(b-a)h^2}{12}\cdot f''(\xi),\ \xi \in]a,b[$$

(Regla compuesta del trapecio).

$$\int_a^b f(x)\,dx \quad = \quad \frac{h}{2}\left(f(a)+f(b)+2\sum_{i=1}^{n-1} f(x_i)\right) - \frac{(b-a)h^2}{12}\cdot f''(\xi) \qquad (5.6)$$

con $\xi \in]a,b[$, $h = \dfrac{b-a}{n}$ y $x_i = a + i\cdot h,\ i = 0,1,2,...,n.$

Ejemplo 5.2

Aunque sabemos que $\displaystyle\int_0^{\pi} \operatorname{sen}(x)\,dx = 2,$ vamos a usar esta integral para ver como funciona la regla compuesta del trapecio. Aproximar $\displaystyle\int_0^{\pi} \operatorname{sen}(x)\,dx$ con tres subintervalos y estimar el error. Además determinar n tal que el error sea $|E| \leq 0.5\times 10^{-8}$.

Solución: $n=3,\ h = \dfrac{\pi-0}{3} = \dfrac{\pi}{3},\ x_0 = 0, x_1 = \dfrac{\pi}{3}, x_2 = \dfrac{2\pi}{3}$ y $x_3 = \pi$. Entonces,

$$\int_0^{\pi} \operatorname{sen}(x)\,dx \approx \frac{\pi/2}{3}\left[\operatorname{sen}(0)+\operatorname{sen}(\pi)+2\cdot(\operatorname{sen}(\pi)+\operatorname{sen}(2\pi/3))\right] = 1.813799364234...$$

El error estimado $|E|$, en valor absoluto, es $\leq \dfrac{\pi\cdot(\pi/3)^2}{12}M$ donde M es el máximo absoluto de $|f''(x)|$ en $[0,\pi]$. En este caso $M=1$ y entonces $|E| \leq 0.287095...$

Ejemplo 5.2 (continuación).

Para determinar n tal que $|E| \leq 0.5 \times 10^{-8}$. Procedemos así: Sabemos que el máximo absoluto de f'' en $[0,\pi]$ es $M = 1$, entonces

$$|E| \leq \frac{(b-a)h^2}{12} \cdot M = \pi \cdot \frac{(\pi/n)^2}{12} = \frac{\pi^3}{12n^2}$$

Como queremos $|E| \leq 0.5 \times 10^{-8}$, basta con que $\dfrac{\pi^3}{12n^2} \leq 0.5 \times 10^{-8}$. Despejando obtenemos,

$$n \geq \sqrt{\frac{\pi^3}{12 \cdot 0.5 \times 10^{-8}}} \approx 22732.603$$

Tomando $n = 22732$, obtenemos la aproximación $\int_0^\pi \operatorname{sen}(x)\,dx \approx 1.99999999681673$, que efectivamente tiene ocho decimales exactos.

EJERCICIOS

5.3 Sea $f \in C^2[a,b]$ y $\xi_k \in]a,b[$, $k = 0,1,...,n-1$. Muestre que existe $\xi \in]a,b[$ tal que

$$\frac{\displaystyle\sum_{k=0}^{n-1} f''(\eta_k)}{n} = f''(\xi)$$

5.4 Si $|f^{(n+1)}(x)| < M$ en $[-1,1]$, estime el error al aproximar $\int_{-1}^{1} f(x)\,dx$ con $\int_{-1}^{1} P_n(x)\,dx$ donde $P_n(x)$ es el polinomio que interpola a f en los $n+1$ nodos de TChebyshev $x_i = \cos\left(\dfrac{2i+1}{2n+2}\pi\right)$, $i = 0,1,....n$.

5.5 La función de Bessel de orden cero se define como

$$J_0(x) = \frac{1}{\pi}\int_0^\pi \cos(x \operatorname{sen} t)\,dt$$

Derivando bajo el signo integral obtenemos

$$\begin{aligned} J_0'(x) &= -\frac{1}{\pi}\int_0^\pi \operatorname{sen} t \operatorname{sen}(x \operatorname{sen} t)\,dt \\ J_0''(x) &= -\frac{1}{\pi}\int_0^\pi (\operatorname{sen} t)^2 \cos(x \operatorname{sen} t)\,dt \\ &\vdots \qquad \vdots \end{aligned}$$

a) Muestre que $|J_0^{(n)}(x)| \leq \dfrac{1}{\pi}\int_0^\pi 1\,dt = 1$, $n = 0,1,2,...$

b) Dado $\delta > 0$, determine n de tal manera que si aproximamos $J_0(x)$ con la regla compuesta del trapecio, el error sea $\leq \delta$.

5.6 Muestre que la regla del trapecio es exacta si f es un polinomio de grado 1.

5.7 Considere la integral $I = \int_1^2 \frac{e^x}{x}\,dx$.

a) Aproximar la integral con la regla compuesta del trapecio con $n = 4$

b) Estime el error en la aproximación anterior.

c) Estime n de tal manera que, usando la regla del trapecio, el error estimado de la aproximación sea $\leq 0.5 \times 10^{-10}$.

5.8 Considere la integral $I = \int_0^1 e^{-x^2}\,dx$.

a) Aproximar la integral con la regla compuesta del trapecio con $n = 4$

b) Estime el error en la aproximación anterior.

c) Estime n de tal manera que, usando la regla del trapecio, el error estimado de la aproximación sea $\leq 0.5 \times 10^{-10}$.

5.9 De una función f, conocemos la siguiente información

x	$f(x)$
0	3.5
0.2	3.1
0.4	3
0.6	2.8
0.8	2.6

Aproximar $\int_0^{0.8} f(x)\,dx$ usando regla del trapecio con $n = 4$.

5.4 Trapecio: Algoritmo e Implementación.

La implementación la hacemos con base en el cuaderno de la figura

	A	B	C	D	E
1	Integración: Regla del Trapecio				
2					
3	Digite				
4	f(x) =	sin(x)			Calcular
6	N		a	b	Trapecio
7	22732		0	3	1,999999997
8					
9					

Figura 5.2

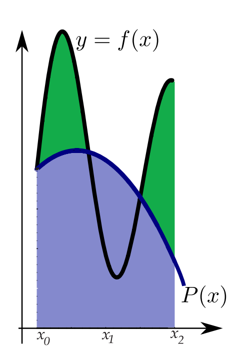


Software: Cuadernos LibreOffice y Excel

Código VBA 5.1: Cuaderno Excel para el método del Trapecio

```
Private Sub CommandButton1_Click()
Call trapecio
End Sub
Sub trapecio()
Dim N As Integer
Dim Formula As String
Dim estaBien As Boolean
Dim Fun As New clsMathParser 'Evaluador
'limpiar celda del resultado
Cells(7, 5) = ""
'intervalo de integración
a = Cells(7, 3)
b = Cells(7, 4)
N = Cells(7, 1)
'leer fórmula f(x) desde el campo de texto
Formula = txtBox1.Text
estaBien = Fun.StoreExpression(Formula)
If Not estaBien Then
MsgBox ("Error " + Fun.ErrorDescription)
Exit Sub
End If
   suma = 0
   h = (b - a) / N
   For i = 1 To N - 1
      xi = a + i * h
      suma = suma + Fun.Eval1(xi)
   Next i
   suma = h / 2 * (Fun.Eval1(a) + Fun.Eval1(b) + 2 * suma)
   Cells(7, 5) = suma
End Sub
```

5.5 Regla del Simpson.



En vez de usar interpolación lineal, usamos interpolación cuadrática buscando una mejora en el cálculo. Por simplicidad, vamos a hacer el análisis en el intervalo $[x_0, x_2]$. Para construir la parábola que interpola f necesitamos los puntos x_0, x_1 y x_2.

Sea $f^{(4)}$ continua en $[a,b]$. Interpolando f en $[a,b]$ con $x_0 = a$, $x_1 = (b+a)/2$ y $x_2 = b$ obtenemos el polinomio de Lagrange $P_2(x)$. Entonces,

$$f(x) = P_2(x) + f[x_0, x_1, x_2, x](x - x_0)(x - x_1)(x - x_2)$$

Luego,

$$\begin{aligned}\int_a^b f(x)\,dx &= \int_a^b P_2(x) + \ (x - x_0)(x - x_1)(x - x_2) f^{(4)}(\xi(x))/2\,dx \\ &= \frac{h}{3}[f(a) + 4f(x_1) + f(b)] \ + \int_a^b (x - x_0)(x - x_1)(x - x_2) f^{(4)}(\xi(x))/2\,dx\end{aligned}$$

En este caso, $\displaystyle\int_a^b (x - x_0)(x - x_1)(x - x_2)\,dx = 0$. Tomando $x_3 = x_1$, el polinomio $Q(x) = (x-a)\left(x - \frac{(a+b)}{2}\right)^2 (x-b)$ es de un mismo signo sobre $[a,b]$. Aplicando el teorema (5.3) tenemos

$$\begin{aligned}\int_a^b f(x)\,dx &= \frac{h}{3}[f(a) + 4f(x_1) + f(b)] \ + \frac{f^{(4)}(\eta)}{4!}\int_a^b Q(x)\,dx, \ \ \eta \in]a,b[\\ &= \frac{h}{3}[f(a) + 4f(x_1) + f(b)] \ - \frac{f^{(4)}(\eta)}{90}\left(\frac{b-a}{2}\right)^5, \ \ \eta \in]a,b[\end{aligned}$$

pues $\displaystyle\int_a^b Q(x)\,dx = -\frac{4}{15}\left(\frac{b-a}{2}\right)^5$.

Para obtener la regla compuesta de Simpson necesitamos n **par** para poder escribir

$$\int_a^b f(x)\,dx = \int_{x_0}^{x_2} f(x)\,dx + \int_{x_2}^{x_4} f(x)\,dx + \ldots + \int_{x_{n-2}}^{x_n} f(x)\,dx$$

Luego calculamos cada una de las $n/2$ integrales:

$$\int_{x_k}^{x_{k+2}} f(x)\,dx = \frac{h}{3}[f(x_k) + 4f(x_{k+1}) + f(x_{k+2})] \ - \frac{f^{(4)}(\eta_k)}{90} h^5$$

con $\eta_k \in]x_k, x_{k+2}[$ y $h = (x_{k+2} - x_k)/2$.

$$\begin{aligned}\int_a^b f(x)\,dx &= \int_{x_0}^{x_2} f(x)\,dx + \int_{x_2}^{x_4} f(x)\,dx + \ldots + \int_{x_{n-2}}^{x_n} f(x)\,dx \\ &= \frac{h}{3}\left[f(a) + f(b) + 4\sum_{i=1}^{n/2} f(x_{2i-1}) + 2\sum_{i=1}^{n/2-1} f(x_{2i})\right] \ - \sum_{k=1}^{n/2}\frac{f^{(4)}(\eta_k)}{90} h^5\end{aligned}$$

con $\eta_k \in]x_k, x_{k+2}[$ y $h = (b-a)/n$.

(Regla compuesta de Simpson).

Si n es par,

$$\begin{aligned}\int_a^b f(x)\,dx &= \int_{x_0}^{x_2} f(x)\,dx + \int_{x_2}^{x_4} f(x)\,dx + \ldots + \int_{x_{n-2}}^{x_n} f(x)\,dx \\ &= \frac{h}{3}\left[f(a)+f(b)+4\sum_{i=1}^{n/2} f(x_{2i-1})+2\sum_{i=1}^{n/2-1} f(x_{2i})\right] - \frac{1}{180}(b-a)h^4 f^{(4)}(\xi)\end{aligned}$$

con $\xi \in]a,b[$, $h = \dfrac{b-a}{n}$ y $x_i = a + i\cdot h$, $i = 0,1,2,\ldots,n$.

La simplificación del término de error se hace como se hizo en la regla del Trapecio.

Aunque la regla de Simpson es muy popular en integración numérica (note que con la misma cantidad de evaluaciones obtenemos una aproximación con un error más pequeño) la regla del Trapecio es más eficiente en ciertos situaciones, como cuando trabajamos con polinomios trigonométricos.

Ejemplo 5.3

Aunque sabemos que $\int_0^{\pi} \operatorname{sen}(x)\,dx = 2$, vamos a usar esta integral para ver como funciona la regla de Simpson.

a.) Aproximar $\int_0^{\pi} \operatorname{sen}(x)\,dx$ con $n = 4$ y estimar el error.

b.) Estime n de tal manera que la regla de Simpson aproxime la integral con un error $|E| \leq 0.5 \times 10^{-8}$.

a.) Solución: Como $n = 4$, calculamos x_0, x_1 x_2 x_3 y x_4.

$n = 4 \implies h = \dfrac{\pi - 0}{4} = \dfrac{\pi}{4}$, $x_0 = 0$, $x_1 = \dfrac{\pi}{4}$, $x_2 = \dfrac{\pi}{2}$, $x_3 = \dfrac{3\pi}{4}$ y $x_4 = \pi$ Entonces,

$$\int_0^{\pi} \operatorname{sen}(x)\,dx \approx \frac{\pi/4}{3}\left[\operatorname{sen}(0) + \operatorname{sen}(\pi) + 4\cdot\left(\operatorname{sen}(\pi/4) + \operatorname{sen}(3\pi/4)\right) + 2\cdot\operatorname{sen}(\pi/2)\right] = 2.004559754984\ldots$$

El error estimado $|E|$, en valor absoluto, es $\leq \dfrac{\pi\cdot(\pi/4)^4}{180} M$ donde M es el máximo absoluto de $|f^{(4)}(x)|$ en $[0,\pi]$. En este caso $M = 1$ y entonces $|E| \leq 0.00664105\ldots$

Ejemplo 5.3 (continuación).

b.) Solución:
Sabemos que el máximo absoluto de $f^{(4)}$ en $[0,\pi]$ es $M = 1$, entonces

$$|E| \leq \frac{(b-a)h^4}{180} \cdot M = \frac{\pi^5}{180n^4}$$

Como queremos $|E| \leq 0.5 \times 10^{-8}$, basta con que $\dfrac{\pi^5}{180n^4} \leq 0.5 \times 10^{-8}$. Despejando obtenemos,

$$n \geq \sqrt[4]{\frac{\pi^5}{180 \cdot 0.5 \times 10^{-8}}} \approx 135.79$$

Tomando $n = 136$, obtenemos la aproximación $\displaystyle\int_0^{\pi} \operatorname{sen}(x)\,dx \approx 2{,}00000000316395...$, que efectivamente tiene ocho decimales exactos.

5.6 Simpson: Algoritmo e Implementación.

Notemos que no se requiere que n sea par, podríamos multiplicar por 2 y resolver el problema. Sin embargo vamos a suponer que n es par para tener control. Una manera directa de implementar la regla adaptativa de Simpson es alternar los x_i de subíndice par y los x_j de subíndice impar y multiplicarlos por 4 y 2 respectivamente. Esto se puede hacer en una sola línea.

Algoritmo 5.1: Regla adaptativa de Simpson

Datos: $f(x)$, a, b y $n \in \mathbb{N}$ par.
Salida: Aproximación de $\displaystyle\int_a^b f(x)\,dx$.

suma= 0;
$h = (b-a)/n$;
for $i = 0$ **to**$(n-1)$ **do**
    suma=suma+$f[a + i \cdot h] + 4 \cdot f[a + (i+1) \cdot h] + f[a + (i+2) \cdot h]$;
    $i = i + 2$;
return suma$\cdot h/3.0$

Implementación en Mathematica para n par.

```
Simpson[f[x_], a_, b_, n_] := Module[{h, suma = 0},
   suma = 0.0;
   h = (b - a)/n;
   For[i = 0, i <= (n - 1), i = i + 2,
    suma =
     suma + f[a + i*h] + 4*f[a + (i + 1)*h] + f[a + (i + 2)*h]];
   Print[h*suma/3.0]
   ];
```

```
(*Corrida*)
f[x_] = Sin[x];
Simpson[f, 0, Pi, 4]
2.0045597549844207
```

Implementación en wxMaxima para $n \in \mathbb{N}$ arbitrario.

```
simpson(f,x1,x2,n):=([j,h,suma],
  suma:0.0,
  h:ev((x2-x1)/(2*n),numer),
  for j:0 thru 2*(n-1) step 2 do(
     suma=suma + f(x1+j*h) + 4.0*f(x1+(j+1)*h) + f(x1+(j+2)*h)
  ),
  h*suma/3.0
  )$
```

```
/*Corrida*/
f(x):=sin(x)$
   simpson(f,0,%pi,2);
   2.0045597549844207
```

EJERCICIOS

5.10 Sea $f \in C^4[a,b]$, $\eta_k \in]a,b[$, $k = 1,...,n/2$ y $h = \dfrac{b-a}{n}$. Muestre que existe $\xi \in]a,b[$ tal que

$$\sum_{k=1}^{n/2} \frac{f^{(4)}(\eta_k)}{90} h^5 = \frac{1}{180}(b-a)h^4 f^{(4)}(\xi)$$

5.11 Si $|f^{(n+1)}(x)| < M$ en $[-1,1]$, estime el error al aproximar $\int_{-1}^{1} f(x)\,dx$ con $\int_{-1}^{1} P_n(x)\,dx$ donde $P_n(x)$ es el polinomio que interpola a f en los $n+1$ nodos de TChebyshev $x_i = \cos\left(\dfrac{2i+1}{2n+2}\pi\right)$, $i = 0,1,....n$.

5.12 La función de Bessel de orden cero se define como

$$J_0(x) = \frac{1}{\pi}\int_0^{\pi} \cos(x \operatorname{sen} t)\,dt$$

Derivando bajo el signo integral obtenemos

$$\begin{aligned}
J_0'(x) &= -\frac{1}{\pi}\int_0^{\pi} \operatorname{sen} t \operatorname{sen}(x \operatorname{sen} t)\, dt \\
J_0''(x) &= -\frac{1}{\pi}\int_0^{\pi} (\operatorname{sen} t)^2 \cos(x \operatorname{sen} t)\, dt \\
\vdots \quad & \quad \vdots
\end{aligned}$$

a) Muestre que $|J_0^{(n)}(x)| \leq \frac{1}{\pi}\int_0^{\pi} 1\, dt \;=\; 1,\; n = 0,1,2,...$

b) Dado $\delta > 0$, determine n de tal manera que si aproximamos $J_0(x)$ con la regla compuesta de Simpson, el error sea $\leq \delta$.

c) Implemente en VBA Excel, una función $J_0 Simpson(x,\delta)$ para aproximar $J_0(x)$, usando la regla de Simpson, con un error estimado $\leq \delta$.

d) En una hoja Excel, realice la representación gráfica de $J_0(x)$ con $x \in [-5,5]$.

e) La función $J_0(x)$ tiene un cero x^* en $[2,3]$. Implemente una versión del método de bisección y una versión del método de Newton que operen con la función $J_0 Simpson(x, 0.5 \times 10^{-10})$ y aproxime en cada caso x^* con un error estimado $\leq 0.5 \times 10^{-8}$.

5.13 Muestre que la regla de Simpson es exacta si f es un polinomio de grado 3.

5.14 Considere la integral $I = \int_1^2 \frac{e^x}{x}\, dx$.

a) Aproximar la integral con la regla compuesta de Simpson con $n = 4$

b) Estime el error en la aproximación anterior.

c) Estime n de tal manera que, usando la regla de Simpson, el error estimado de la aproximación sea $\leq 0.5 \times 10^{-10}$.

5.15 Considere la integral $I = \int_0^1 e^{-x^2}\, dx$.

a) Aproximar la integral con la regla compuesta de Simpson con $n = 4$

b) Estime el error en la aproximación anterior.

c) Estime n de tal manera que, usando la regla comuesta de Simpson, el error estimado de la aproximación sea $\leq 0.5 \times 10^{-10}$.

5.16 De una función f, conocemos la siguiente información

x	$f(x)$
0	3.5
0.2	3.1
0.4	3
0.6	2.8
0.8	2.6

Aproximar $\int_0^{0.8} f(x)\, dx$ usando regla compuesta de Simpson con $n = 4$.

5.7 Método de Romberg.

El método de Romberg usa la regla compuesta del trapecio para obtener aproximaciones preliminares y luego el proceso de extrapolación de Richardson para mejorar las aproximaciones.

5.7.1 Extrapolación de Richardson.

Supongamos que queremos estimar I y que podemos expresar I como

$$I = T(h) + a_2h^2 + a_4h^4 + a_6h^6 + \cdots \tag{5.7}$$

En este caso $T(h)$ es una aproximación de I y $a_4h^4 + a_6h^6 + \cdots$ es el error de la estimación.

Supongamos además que T solo se puede evaluar para $h > 0$ (sino, el error sería nulo y no habría nada qué hacer) por lo que lo único que podemos hacer es tomar valores cada vez más pequeños de h.

Si $h \longrightarrow 0$ entonces las potencias $h^4, h^6, ...$ se hacen pequeñas rápidamente por lo que, en la expresión del error $a_4h^4 + a_6h^6 + \cdots$, el sumando que aporta la mayor parte del error es a_4h^4 (si $a_4 \neq 0$). En un primer paso, el método de Romberg pretende mejorar la estimación de L eliminando este sumando. Para hacer esto procedemos así: En la ecuación (5.7) sustituimos h por $h/2$, luego podemos eliminar el sumando h^2 multiplicando está última expresión por 4 y restando la expresión (5.7):

$4I = 4T(h/2) \; + \; 4a_2h^2/2^2 \; + \; 4a_4h^4/2^4 \; + \; 4a_6h^6/2^6 + \cdots$

$-I = -T(h) \; - \; a_2h^2 \; - \; a_4h^4 \; - \; a_6h^6 + \cdots$

Sumando y despejando obtenemos

$$I = \frac{4}{3}T(h/2) \; - \; \frac{1}{3}T(h) \; - \; a_4h^4/2^2 \; - \; 5a_6h^6/16 + \cdots \tag{5.8}$$

¿Cuál es la ganancia? I ahora se aproxima con $\frac{4}{3}T(h/2) \; - \; \frac{1}{3}T(h)$ con un error más pequeño: $- \; a_4h^4/2^2 \; - \; 5a_6h^6/16 + \cdots$.

Usando la notación $O-$grande diríamos que en (5.7) el error es de orden $O(h^2)$ (pues h^2 es la potencia dominante) mientras que en (5.8) expresión el error es de orden $O(h^4)$.

Ahora aplicamos el mismo procedimiento a $I = T_1(h) \; - \; b_4h^4 \; - \; b_6h^6 + \cdots$, con $T_1(h) = 4/3\,T(h/2) \; - \; 1/3\,T(h)$, $b_4 = a_4/4$, $b_6 = a_6/2^3$...

Para eliminar b_4h^4 cambiamos h por $h/2$ y multiplicamos por 16 y restamos la ecuación inicial

$16I = 16T_1(h/2) \; - \; 16b_4h^4/2^4 \; - \; 16b_6h^6/64 + \cdots$

$-I = -T_1(h) \; - \; b_4h^4 \; - \; b_6h^6 + \cdots$

Sumando y despejando obtenemos

$$I = \frac{16}{15}T_1(h/2) \; - \; \frac{1}{15}T_1(h) \; - \; b_6 h^6/20 + \cdots$$

Poniendo $T_2(h) = \frac{16}{15}T_1(h/2) \; - \; \frac{1}{15}T_1(h)$, entonces podemos decir que $T_2(h)$ aproxima I con un error de orden $O(h^6)$.

Siguiendo este procedimiento obtenemos $I = \frac{4^k}{4^k-1}T_k(h/2) \; - \; \frac{1}{4^k-1}T_k(h) + O(h^{2(k+1)})$ en el paso $k-$ésimo. Esto se puede simplificar y poner como

$$I = T_k(h/2) \; + \; \frac{T_k(h/2) - T_k(h)}{4^k - 1} + O(h^{2(k+1)})$$

5.8 Método de Romberg

El método de Romberg es una aplicación sistemática de esta idea de obtener una aproximación mejorada a partir de aproximaciones anteriores, iniciando con estimaciones de la regla del Trapecio para $h_k = \frac{b-a}{2^{k-1}}, k = 1, 2, ...$

Sea $h_k = \frac{b-a}{2^{k-1}}$, $I = \int_a^b f(x)\,dx$ y $T(h) = \frac{h}{2}\left(f(a) + f(b) + 2\sum_{i=1}^{n-1} f(x_i)\right)$. Para aplicar el proceso de extrapolación de Richardson necesitamos

$$I = T(h) + a_2h^2 + a_4h^4 + a_6h^6 + \cdots + a_{2m-2}h^{2m-2} + a_{2m}h^{2m}f^{(2m)}(\varepsilon) \tag{5.9}$$

Esto es cierto, pero para justificarlo necesitamos la fórmula de Euler-Maclaurin ([1]), así que aquí vamos a asumir este hecho.

El método de Romberg construye una matriz $R = (R_{i,j})$ en la que todas sus columnas convergen a I (las entradas son sumas de Riemann) pero la rapidez de convergencia crece de una columna a otra y esto se logra usando extrapolación de Richardson.

$$\begin{array}{cccccc}
\text{Trapecio} & \text{Extrapolación} & \text{Extrapolación} & & & \\
R_{1,1} & & & & & \\
R_{2,1} & R_{2,2} & & & & \\
R_{3,1} & R_{3,2} & R_{3,3} & & & \\
\vdots & \vdots & \vdots & \vdots & \ddots & \\
R_{n,1} & R_{n,2} & R_{n,3} & R_{n,4} & \cdots & R_{n,n} \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
I & I & I & I & I & I
\end{array}$$

La primera columna de la matriz son los resultados de aplicar regla compuesta del trapecio: Se elige $h = b - a$ y se aplica regla del trapecio con $h_k = \dfrac{h}{2^{k-1}}, k = 1, 2, ...$

Ejemplo 5.4

Vamos a calcular $\begin{array}{ll} R_{1,1} & \\ R_{2,1} & R_{2,2} \end{array}$

Cálculo de $R_{1,1}$: $h_1 = h = (b - a)$,

$$R_{1,1} = T(h) = \frac{b-a}{2}\left[f(a) + f(b)\right]$$

Cálculo de $R_{2,1}$: $h_2 = h/2 = (b - a)/2$,

$$\begin{aligned} R_{2,1} = T(h/2) &= \frac{h}{2}\left(f(a) + f(b) + 2\sum_{i=1}^{2^1-1} f(x_i)\right) \\ &= \frac{(b-a)}{4}\left[f(a) + f(b) + 2f\left(\frac{a+b}{2}\right)\right] \end{aligned}$$

Extrapolación:

$$\begin{aligned} R_{2,2} &= 4/3\,T(h/2) \;-\; 1/3\,T(h) \\ &= \frac{4}{3}\frac{(b-a)}{4}\left[f(a) + f(b) + 2f\left(\frac{a+b}{2}\right)\right] - \frac{1}{3}\frac{b-a}{2}\left[f(a) + f(b)\right] \\ &= \frac{b-a}{6}\left(f(a) + f(b) + 4f\left(\frac{a+b}{2}\right)\right) \end{aligned}$$

Recordemos que $R_{2,2} = R_{2,1} + \dfrac{R_{2,1} - R_{1,1}}{4^{2-1} - 1}$

La notación $R_{k,1}$ corresponde a la aproximación por Trapecios.

- $R_{1,1} = \dfrac{h_1}{2}\left[f(a) + f(b)\right] = \dfrac{b-a}{2}\left[f(a) + f(b)\right]$

- $R_{k,1} = \frac{1}{2}\left[R_{k-1,1} + h_{k-1}\sum_{i=1}^{2^{k-2}} f[a+(2i-1)\cdot h_k]\right],\ k = 2,3,...,n.$

Observe que $R_{k,1}$ es un fórmula recursiva para la regla compuesta del trapecio.

En particular,

$$R_{1,1} = \frac{b-a}{2}\,[f(a)+f(b)]$$

$$R_{2,1} = \frac{(b-a)}{4}\left[f(a)+f(b)+2f\left(\frac{a+b}{2}\right)\right]$$

$$R_{3,1} = \frac{1}{2}\left[R_{2,1} + \frac{a+b}{2}\left[f\left(\frac{3a+b}{4}\right)+f\left(\frac{a+3b}{4}\right)\right]\right]$$

Luego, haciendo $h_{k+1} = h_k/2$ podemos obtener nuevas aproximaciones $R_{k,k}$ de manera recursiva (usando extrapolación), de la siguiente manera,

$$R_{k,j} = R_{k,j-1} + \frac{R_{k,j-1} - R_{k-1,j-1}}{4^{j-1}-1}$$

En particular, si en la expansión (5.9) cada $a_i \neq 0$, entonces

$$R(n,m) = I + O\left(\frac{1}{2^{(n-1)(m-1)}}\right)$$

El cálculo se hace sencillo si formamos la matriz

$$\begin{array}{cccccc}
R_{1,1} \\
R_{2,1} & R_{2,2} \\
R_{3,1} & R_{3,2} & R_{3,3} \\
R_{4,1} & R_{4,2} & R_{4,3} & R_{4,4} \\
\vdots & \vdots & \vdots & \vdots & \ddots \\
R_{n,1} & R_{n,2} & R_{n,3} & R_{n,4} & \cdots & R_{n,n}
\end{array}$$

Observe que el esquema de cálculo es similar al de diferencias divididas de Newton.

5.8.1 Algoritmo e Implementación en VBA Excel.

Podemos usar $\text{Mín}_k\{R_{n,k} - R_{n,k-1}\}$ (el mínimo de las restas de cada dos columnas consecutivas en la fila n) como un estimado del error de truncación aunque frecuentemente éste resulta en una sobreestimación.

En la implementación del método de Romberg podríamos usar como criterio de parada: calcular la fila $n > 1$ hasta que $\text{Mín}_k \{R_{n,k} - R_{n,k-1}\} < \delta$ o $n \leq$ `numIter`. Aquí `numIter` no debería ser más grande que, digamos 15. Además se debe indicar un número mínimo de iteraciones, digamos $n = 3$. Se trata de una heurística para evitar la finalización prematura cuando el integrando oscila mucho.

Ejemplo 5.5

Aunque sabemos que $\int_0^{\pi} \text{sen}(x)\,dx = 2$, vamos a usar esta integral para ver como funciona la regla de Simpson. Aproximar $\int_0^{\pi} \text{sen}(x)\,dx$ con $n = 4$ y $n = 6$ y estimar el error.

Solución: Calculamos la matriz de Romberg.

k				
1	0			
2	1.570796327	2.094395102		
3	1.896118898	2.004559755	1.998570732	
4	1.974231602	2.00026917	1.999983131	2.00000555

Luego $\int_0^{\pi} \text{sen}(x)\,dx \approx 2.000000016$. La estimación del error de truncación requiere comparar los elementos consecutivos de la última fila y escoger el menor

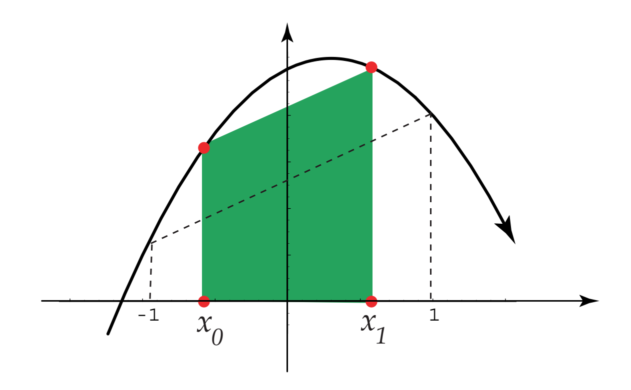
$$|R_{4,1} - R_{4,2}| = 0.0260376 \quad |R_{4,2} - R_{4,3}| = 0.000286039 \quad \text{y} \quad |R_{4,2} - R_{4,3}| = 0.000286039.$$

Así que la estimación del error en la truncación es de ± 0.000022419.

5.9 Cuadratura Gaussiana.

En la cuadratura Gaussiana (método de Gauss para aproximar una integral), en vez de usar una partición igualmente espaciada del intervalo $[a,b]$ para aproximar la integral con $n+1$ puntos, se escogen los "mejores" $x_0, x_1, ... x_n \in [a,b]$, de tal manera que la aproximación sea exacta al menos, para polinomios de grado menor o igual a $2n+1$ (recordemos que Trapecio es exacto para polinomios de grado 1 y Simpson para polinomios de grado 3).

En la figura 5.9, el área de la región que cubre el trapecio relleno es exactamente el área de la región entre la parábola y el eje X. La aproximación que da la regla del Trapecio (trapecio punteado) no es exacta en este caso.



En general, se trata de usar el método de *coeficientes indeterminados:* determinar $c_0, c_1, ..., c_n$ y $x_0, x_1, ..., x_n \in [-1,1]$ de tal manera que las integrales

$$\int_{-1}^{1} P_{2n+1}(x)\,dx = c_0 f(x_0) + c_1 f(x_1) + ... + c_n f(x_n)$$

son exactas para cada $P_{2n+1}(x)$, un polinomio de grado $2n+1$, $n = 0,1,2,...$

Debemos resolver el sistema (no lineal) con $n+1 \ + \ n+1 = 2n+2$ incognitas,

$$\left\{\begin{array}{lclcll} c_0 f(x_0) + c_1 f(x_1) + ... + c_n f(x_n) & = & \int_{-1}^{1} 1\,dx & = & 2, & f(x) = 1 \\ c_0 f(x_0) + c_1 f(x_1) + ... + c_n f(x_n) & = & \int_{-1}^{1} x\,dx & = & 0, & f(x) = x \\ & \vdots & & \vdots & & \\ c_0 f(x_0) + c_1 f(x_1) + ... + c_n f(x_n) & = & \int_{-1}^{1} x^{2n}\,dx & = & \dfrac{2}{2n+1}, & f(x) = x^{2n} \\ c_0 f(x_0) + c_1 f(x_1) + ... + c_n f(x_n) & = & \int_{-1}^{1} x^{2n+1}\,dx & = & 0, & f(x) = x^{2n+1} \end{array}\right.$$

En la tabla que sigue, aparecen la solución aproximada, hasta $n = 5$.

n	c_i	x_i
1	$c_0 = 1$	$x_0 = -0.5773503$
	$c_1 = 1$	$x_1 = -0.5773503$
2	$c_0 = 0.5555555556$	$x_0 = -0.7745966692$
	$c_1 = 0.8888888889$	$x_1 = \ \ 0$
	$c_2 = 0.5555555556$	$x_2 = \ \ 0.7745966692$
3	$c_0 = 0.3478548451$	$x_0 = -0.8611363116$
	$c_1 = 0.6521451549$	$x_1 = -0.3399810436$
	$c_2 = 0.6521451549$	$x_2 = \ \ 0.3399810436$
	$c_3 = 0.3478548451$	$x_3 = \ \ 0.8611363116$
4	$c_0 = 0.2369268850$	$x_0 = -0.9061798459$
	$c_1 = 0.4786286705$	$x_1 = -0.5384693101$
	$c_2 = 0.5688888889$	$x_2 = \ \ 0$
	$c_3 = 0.4786286705$	$x_3 = \ \ 0.5384693101$
	$c_4 = 0.2369268850$	$x_4 = \ \ 0.9061798459$
5	$c_0 = 0.1713245$	$x_0 = -0.932469514$
	$c_1 = 0.3607616$	$x_1 = -0.661209386$
	$c_2 = 0.4679139$	$x_2 = -0.238619186$
	$c_3 = 0.4679139$	$x_3 = \ \ 0.238619186$
	$c_4 = 0.3607616$	$x_4 = \ \ 0.661209386$
	$c_5 = 0.1713245$	$x_5 = \ \ 0.932469514$

Tabla 5.1

Una tabla más extensa se puede encontrar en Proyecto Euclides.
Se puede probar que $x_0, x_1, ... x_n$ son las raíces de los polinomios de Legendre,

$$P_{n+1}(x) = \frac{(n+1)!}{(2n+2)!} G_n^{(n+1)}(x) \ \ n = 1,2,...$$

con $G_n^{(n+1)}(x)$ la derivada $n+1$ de $G_n(x) = (x^2-1)^{n+1}$.

Si tenemos las raíces (que son todas reales), los c_i's se podrían calcular con la fórmula

$$c_i = \int_{-1}^{1} \prod_{j=0, j\neq i}^{n} \frac{(x-x_j)}{(x_i-x_j)}\,dx$$

(Cuadratura Gaussiana).

- Para calcular en un intervalo $[a,b]$ usando cuadratura Gausiana hacemos el cambio de variable $x = \dfrac{a+b+(b-a)u}{2}$ y teniendo en cuenta que $dx = \dfrac{b-a}{2}du$, obtenemos

$$\int_a^b f(x)\,dx = \int_{-1}^{1} \frac{b-a}{2} f\left(\frac{a+b+(b-a)u}{2}\right) du \;=\; c_0\,g(x_0) + c_1\,g(x_1) + ... + c_n\,g(x_n) \;+\; E_n$$

donde, por supuesto, $g(u) = \dfrac{b-a}{2} f\left(\dfrac{a+b+(b-a)u}{2}\right)$.

- Si $g \in C^{2n}[-1,1]$, el error en la fórmula de cuadratura Gaussiana es ([20]),

$$E_n = \frac{2^{2n+1}[(n)!]^4}{(2n+1)[(2n)!]^3}\,g^{(2n)}(\xi) \quad \text{con} \quad \xi \in]-1,1[.$$

Ejemplo 5.6

Aproximar $\displaystyle\int_0^{\pi} \text{sen}(x)\,dx$ con $n=3$ y estimar el error.

Solución:

- En este caso, el cambio de variable es $x = \dfrac{0+\pi+(\pi-0)u}{2} = \dfrac{\pi+\pi u}{2}$ y $dx = \dfrac{\pi}{2}du$.
- $g(u) = \dfrac{\pi}{2}\,\text{sen}\left(\dfrac{\pi+\pi u}{2}\right) = \dfrac{\pi}{2}\cos\left(\dfrac{\pi u}{2}\right)$.

$$\begin{aligned}
\int_0^{\pi} \text{sen}(x)\,dx &= \int_{-1}^{1} \frac{\pi}{2}\cos(\pi u/2)\,du \\
&\approx c_0\,g(x_0) + c_1\,g(x_1) + c_2\,g(x_2) + c_3\,g(x_3) \\
&= \pi/2\left[c_0\cos\left(\frac{\pi x_0}{2}\right) + c_1\cos\left(\frac{\pi x_1}{2}\right) + c_2\cos\left(\frac{\pi x_2}{2}\right) + c_3\cos\left(\frac{\pi x_3}{2}\right)\right] \\
&\quad \text{y usando la tabla de valores,} \\
&= 1.9999842284987...
\end{aligned}$$

Ejemplo 5.6 (continuación).

La estimación del error es,

$$
\begin{aligned}
|E_3| &= \frac{2^7 \cdot (3!)^4}{7 \cdot (6!)^3} \cdot |g^{(6)}(\xi)| \quad \text{con} \quad \xi \in]-1,1[. \\
&\leq \frac{2^7 \cdot (3!)^4}{7 \cdot (6!)^3} \cdot \frac{\pi^7}{128} \approx 0.00149816
\end{aligned}
$$

pues $|g^{(6)}(u)| = \dfrac{\pi^7 \cos\left(\dfrac{\pi u}{2}\right)}{128} \leq \dfrac{\pi^7}{128}$ en $]-1,1[$.

5.10 Integrales Impropias.

Las integrales impropias (convergentes) $\displaystyle\int_a^\infty f(x)\,dx$, $(a > 0)$ y $\displaystyle\int_{-\infty}^b f(x)\,dx$, $(b < 0)$; se pueden calcular usando un cambio de variable.

Si $a > 0$ y $b = \infty$ o si $b < 0$ y $a = -\infty$ entonces,

$$\int_a^b f(x)\,dx = \int_{1/b}^{1/a} \frac{1}{t^2} f\left(\frac{1}{t}\right) dt$$

Como $1/a$ o $1/b$ es cero, $f\left(\dfrac{1}{t}\right)$ se indefine. Así que no podemos considerar métodos de integración que evalúen los extremos (como Simpson o Trapecio) sino más bien, de acuerdo a lo que tenemos hasta aquí, cuadratura Gaussiana.

EJERCICIOS

5.17 Considere la integral $I = \displaystyle\int_0^1 e^{-x}\,dx$.

a) Aproximar la integral con la regla del Trapecio para $n = 4$ y estime el error de la aproximación.

b) Estime n de tal manera que, usando la regla del Trapecio, el error estimado es $\leq 0.5 \times 10^{-10}$. Use su implementación en Excel para calcular la aproximación correspondiente (de I).

c) Aproximar la integral con la regla del Simpson para $n = 4$ y estime el error de la aproximación.

d) Estime n de tal manera que, usando la regla del Simpson, el error estimado sea $\leq 0.5 \times 10^{-10}$. Use su implementación en Excel para calcular la aproximación correspondiente (de I).

e) Aproximar la integral con el método de Romberg para $n=4$ y estime el error de la aproximación.

f) Use su implementación del método de Romberg para encontrar experimentalmente, usando la estimación del error, el n adecuado para el que el error estimado sea $\leq 0.5\times 10^{-10}$.

g) Aproximar la integral con cuadratura Gaussiana para $n=4$ y estime el error de la aproximación.

h) Estime n de tal manera que, usando cuadratura Gaussiana, el error estimado sea $\leq 0.5\times 10^{-10}$.

Ayuda: aquí no se trata de hacer un despeje de n (por la presencia de factoriales) sino, más bien, ensayar (tanteo) con valores de n (en la fórmula del error) hasta lograr el objetivo. Observe que $g(u)=\dfrac{1}{2}\,e^{-(u+)/2}$. Las derivadas de g tienen un patrón:

$$\begin{aligned}
|g'(u)| &= \frac{1}{4}\,e^{-(u+1)/2}\\
|g''(u)| &= \frac{1}{8}\,e^{-(u+1)/2}\\
|g^{(3)}|(u) &= \frac{1}{16}\,e^{-(u+1)/2}\\
|g^{(4)}(u)| &= \frac{1}{32}\,e^{-(u+1)/2}\\
\ldots
\end{aligned}$$

5.18 Se conoce que $\displaystyle\int_0^1 \frac{\cos x}{\sqrt{x}}\,dx = 1.80904...$ Aproxime esta integral usando $n=4$.

5.19 Considere las integrales de Fresnel, $S(x)=\displaystyle\int_0^x \operatorname{sen}(t^2)\,dt$ y $Cx)=\displaystyle\int_0^x \cos(t^2)\,dt$. Se sabe que existe un valor $\xi\in[1,2]$ tal que $S(\xi)=C(\xi)$. Aproxime este valor usando el método de la secante.

5.20 La "función error" se define como $\operatorname{Erf}(x)=\dfrac{2}{\sqrt{\pi}}\displaystyle\int_0^x e^{-t^2}\,dt$. Aproximar $\operatorname{Erf}(1.5)$ usando los cuatro métodos de integración hasta que la diferencia en cada resultado sea $\leq 0.5\times 10^{-5}$.

5.21 De una función f, conocemos la siguiente información

x	$f(x)$
0	3.592
0.2	3.110
0.4	3.017
0.6	2.865
0.8	2.658

Tabla 5.2

a) Aproximar $\displaystyle\int_0^{0.8} f(x)\,dx$ usando regla del Trapecio.

b) Aproximar $\int_0^{0.8} f(x)\,dx$ usando regla del Simpson.

c) Aproximar $\int_0^{0.8} f(x)\,dx$ usando Romberg (interpolar con polinomios de grado 2).

5.22 Aproxime $\int_1^{\infty} \frac{1}{\sqrt{2\pi}} e^{-x^2/2}\,dx$ con $n = 6$.

5.23 Aproxime $\int_1^{5} \frac{1}{\sqrt{2\pi}} e^{-x^2/2}\,dx$ con $n = 6$.



6 ECUACIONES DIFERENCIALES ORDINARIAS

Consideremos el problema de valor inicial

$$\frac{dy}{dt} = f(t, y(t)), \quad a \leq t \leq b, \quad y(a) = y_0 \tag{6.1}$$

Buscamos una función $y(t) \in c^1[a,b]$ que satisfaga la identidad (6.1). Se asume que $f(t,y)$ está definida para $t \in [a,b]$ y $y \in \mathbb{R}^m$. Por supuesto, en este texto solo estudiamos el caso $m = 1$.

Existencia y unicidad. En teoría de ecuaciones diferenciales se establece el siguiente teorema,

Teorema 6.1

Si $f(t,y)$ es continua en $t \in [a,b]$ y respecto a y satisface la condición de Lipschitz

$$||f(t,y) - f(t,y^*)|| \leq L||y - y^*||, \quad t \in [a,b], \; y, \, y^* \in \mathbb{R},$$

entonces el problema de valor inicial (6.1) tiene una única solución $y(t)$, $a \leq t \leq b$, para cualquier $y_0 \in \mathbb{R}$.

Supongamos que tenemos el problema de valor inicial

$$\frac{dy}{dt} = f(t,y), \quad a \leq t \leq b, \quad y(a) = y_0 \tag{6.2}$$

Si tenemos una aproximación (t_i, y_i) de $(t_i, y(t_i))$, el paso siguiente en un método de un solo paso es

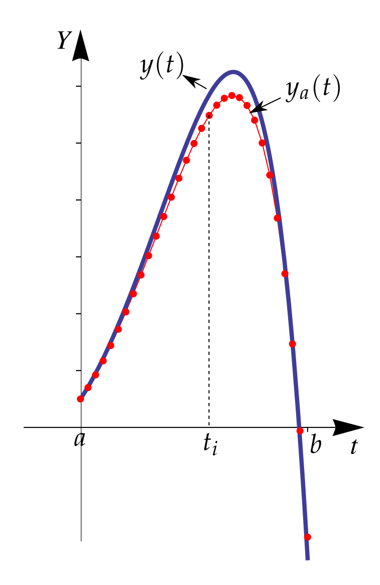
$$y_{i+1} = y_i + h \cdot \Phi(t_i, y_i; h), \quad h > 0.$$

La función Φ se puede ver como el incremento aproximado en cada paso y define cada método de un solo paso.

Orden del método. Para definir el orden del método necesitamos definir el *error de truncación*. Sea $u(t)$ la solución del problema 6.2 pero pasando por el punto (genérico) (x,y), es decir, $u(t)$ es la solución del problema local

$$\frac{du}{dx} = f(t,u), \quad x \leq t \leq x + h, \quad u(x) = y \tag{6.3}$$

A $u(t)$ se le llama *solución de referencia*. Si $y^* = y + h\Phi(x,y;h)$, y^* aproxima $u(x+h)$ con un *error de truncación* $T(x,y;h) = \dfrac{1}{h}(y^* - u(x+h))$.



Solución numérica. Desde el punto de vista numérico lo que nos interesa encontrar aproximaciones $y_a(t_i)$ a los valores exactos $y(t_i)$. En este capítulo, los $t_i \in [a,b]$ los tomaremos igualmente espaciados, es decir, si $h = (b-a)/n,\ ti = a + i \cdot h,\ i = 0,1,...,n.$

Si uno lo prefiere, puede construir una tabla de aproximaciones $\{(t_i, y_a(ti)),\ i = 0,1,...,n\}$ y por interpolación, contruir una solución aproximada $y_a(t)$, $t \in [a,b]$.

Figura 6.1 Solución numérica de un problema de valor inicial

El método Φ se dice de orden p si $||T(x,y;h)|| \leq Ch^p$ uniformenente sobre $[a,b]$ donde la constante C no depende de x,y o h. Esta propiedad es usual escribirla como

$$T(x,y;h) = O(h^p),\ \ h \to 0,$$

es decir, entre más grande p, más exacto es el método.

A continuación, vamos a ver algunos métodos de un solo paso.

6.1 Método de Euler

Euler propuso este método en 1768. Consiste en seguir la tangente en cada punto (t_i, y_i). Hacemos una partición del intervalo $[a,b]$ en n subintervalos $[t_i, t_{i+1}]$, cada uno de longitud $h = (b-a)/n$. Luego, $t_{i+1} = a + i \cdot h = t_i + h$. Iniciando con (t_0, y_0), se calcula la ecuación de la tangente en (t_i, y_i): $y_T(t) = f(t_i, y_i)(t - t_i) + y_i$ y se evalúa en $t = t_{i+1} = t_i + h$, es decir,

$$y(t_{i+1}) \approx y_{i+1} = y_i + h\,f(t_i, y_i),\ \ \ i = 0,1,...,n$$

El método de Euler es de orden $p = 1$.

6.2 Algoritmo e implementación con WXMAXIMA.

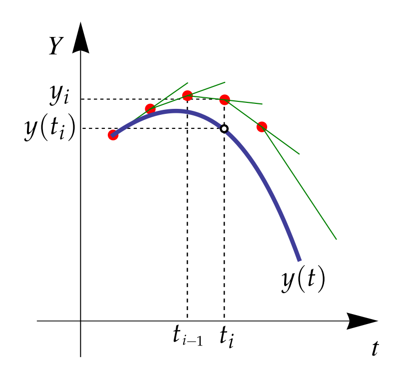


Figura 6.2 $y(t_i) \approx y_i = y_{i-1} + h f(t_{i-1}, y_i)$.

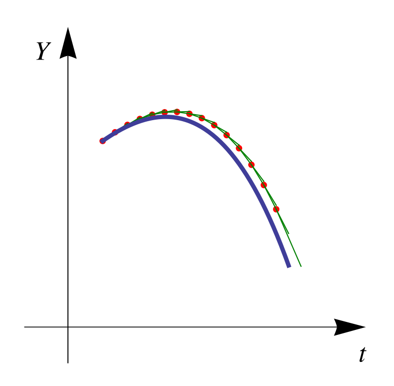


Figura 6.3 Tangentes en (t_i, y_i), $i = 0, 1, ...15$.

Ejemplo 6.1

Consideremos el problema de valor inicial $\dfrac{dy}{dt} = 0.7y - t^2 + 1$, $t \in [1,2]$, $y(1) = 1$. Aquí $a = 1, b = 2$. Si $n = 10$ entonces $h = 0.1$ y $t_i = a + hi = 1 + 0.1i$

$$\begin{cases} y_0 & = & 1 \\ y_{i+1} & = & y_i + h\,(0.7 * y_i - t_i{}^2 + 1) = y_i + 0.1\left(0.7y_i - (1 + 0.1i)^2 + 1\right), \quad i = 1, ..., n \end{cases}$$

i	t_i	y_i
0	1	1
1	1.1	1.07
2	1.2	1.1239
3	1.3	1.15857
4	1.4	1.17067
5	1.5	1.15662
6	1.6	1.11258
7	1.7	1.03446
8	1.8	0.917877
9	1.9	0.758128
10	2.	0.550197

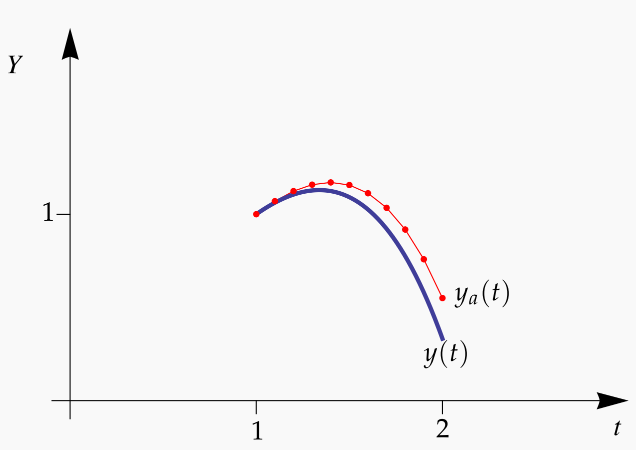


Tabla 6.1 $y(t_i) \approx y_i = y_{i-1} + h f(t_{i-1}, y_i)$, $i = 1, 2, ..., 10$.

En el algoritmo se usan las variables t_0, y_0 y y_1. En cada ciclo t_0 y y_0 se actualizan para calcular la siguiente aproximación $y_1 = f(t_0, y_0) \cdot h + y0$. Luego se incrementa t_0 y se actualiza $y_0 = y_1$.

Algoritmo 6.1: Método de Euler

Datos: $f(t,y)$, a, b, y_0, n
Salida: Imprime las aproximaciones (t_i,y_i), $i = 0,1,...,n$

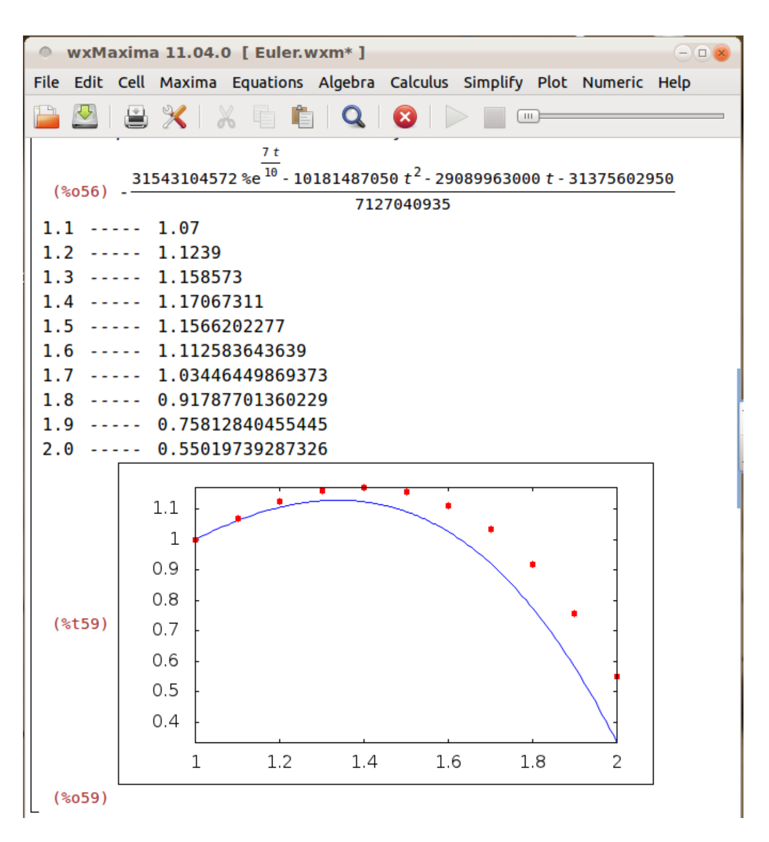
$h = (b-a)/n$;
$t_0 = a$;
for $i = 1$ **to** n **do**
  $y_1 = y_0 + h \cdot f(t_0,y_0)$;
  $t_0 = a + i \cdot h$;
  $y_0 = y_1$;
  print((t_0,y_0));

Implementación en wxMaxima. En esta primera implementación, definimos la función $f(t,y)$ y luego la llamamos en el programa por su nombre `'f'`.

```
f(t,y):=0.7*y-t^2+1; /* dy/dt = f(t,y) */
Euler(f,a,b,y00,n):=block([h:(b-a)/n*1.0,y0:y00,y1,t0],
print(a,"------",y0*1.0),
t0:a,
for  i:1 thru n do (
        y1:y0+h*f(t0,y0),
        /*Actualizamos t0 y y0*/
        t0:a+i*h,
        y0:y1,
        print(t0,"------",y0*1.0)
                   )
)%$
/*Llamada al programa*/
Euler(f,1,2,1,10);
```

```
/* Salida */
1 ------ 1.0
1.1 ------ 1.07
1.2 ------ 1.1239
1.3 ------ 1.158573
1.4 ------ 1.17067311
1.5 ------ 1.1566202277
1.6 ------ 1.112583643639
1.7 ------ 1.03446449869373
1.8 ------ 0.91787701360229
1.9 ------ 0.75812840455445
2.0 ------ 0.55019739287326
```

Segunda implementación. Esta segunda versión tiene fines didácticos. Se acepta la expresión 'fty' sin importar el nombre de las variables. Si las variables son 't' y 'y', en ese orden,debemos hacer la sustitución de variables, usando 'substitute' y luego definir la función 'f(t,y)'. Se usa el paquete 'draw2d' para desplegar los gráficos: Una lista con los puntos (t_{i+1}, y_{i+1}) y la solución exacta (explícita) $y = y(t)$ de la ecuación diferencial. La salida esperada del programa será




```
/*Versión para uso didáctico*/
/*Se supone que la EDO tiene solución expl\'{\i}cita y=y(t)*/
eq:'diff(y,t) = 0.7*y-t^2+1;
sol:ode2(eq, y, t);
yt:rhs(ic1(sol,t= 1.0,y= 1.0));/*Parte derecha*/
load("draw");

PlotEuler(fty,var1,var2,a,b,y00,n):=block(
    [h:1.0*(b-a)/n,y0:y00,y1,t0,tiyi],
    subst(t,var1,fty),
    subst(y,var2,fty),
    define(f(t,y),fty),
    t0:a,
    tiyi:[[t0,y0]], /*matriz inicia en (t0,y0)*/

    for  i:1 thru n do (
        y1:y0+h*f(t0,y0),
```

```
            /*Actualizar t0 y y0 */
        t0:a+i*h,
        y0:y1,
           /*agregar nuevo punto*/
        tiyi:append(tiyi,[[t0,y0]]),

        print(t0," ----- ",y0)
                  ),
    /*gráfico*/
    wxdraw2d(
        explicit(yt,t,a,b),
        color=red, point_size=1,
        point_type=filled_circle,
        points(tiyi)
        /*,yrange=[0,8], xrange=[-1,5]*/
            )
);
/*Llamada*/
PlotEuler(0.7*y-t^2+1,t,y,1,2,1,10);
```

Si la ecuación diferencial tiene solución implícita entonces se debe cambiar el código en la parte del cálculo de la solución de la ecuación diferencial y el gráfico, porque se debe graficar ahora una solución implícita. Por ejemplo, la ecuación diferencial $\dfrac{dy}{dt} = t/y$ con $y(2) = -1$ tiene solución implícita $\dfrac{y^2}{2} = \dfrac{t^2-3}{2}$. Habría que hacer el siguiente ajuste,

```
...
/*Solución y(t)*/
eq:'diff(y,t) = f(t,y),
sol:ode2(eq, y, t),
yt:ic1(sol,t= a,y= y00),/*Tomamos toda la expresión*/
print("------    ",yt),
/*gráfico*/
wxdraw2d(
implicit(yt,t,0,b,y,-3,3),  /*yt es una expresióm impl\'{\i}cita*/
color=red, point_size=1, point_type=filled_circle,points(tiyi),
yrange=[-4,4], xrange=[0,5]
)
);
/*Llamada*/
PlotEuler(t/y,t,y,2,3,-1,10);
```

Por supuesto, este programa sirve también para el caso explícito.

6.3 Métodos de Taylor de orden superior.

El método de Euler opera con un polinomio de Taylor de orden uno (rectas). Es natural, como propuso Euler, usar más términos e la expansión de Taylor (si f es suficientemente derivable). Usando una expansión de orden m nos lleva a un método de orden $O(h^m)$. El costo de calcular las derivadas en estos tiempos se le delega a los computadores, lo que hace que el método (todavía de un solo paso), sea una opción viable.

En este método, calculamos el polinomio de Taylor alrededor de $t = t_i$ (en potencias de $t - t_i$) y evaluamos este polinomio en $t_{i+1} = t_i + h$. Nos queda un polinomio en potencias de h,

$$y(t_{i+1}) = y(t_i + h) = y(t_i) + hy'(t_i) + \frac{h^2}{2}y''(t_i) + ... + \frac{h^m}{m!}y^{(m)}(t_i) + \frac{h^{m+1}}{(m+1)!}y^{(m+1)}(\xi_i), \quad \text{con} \quad \xi_i \in]t_i, t_i + h[.$$

Como $y'(t) = f(t,y)$, las derivadas sucesivas se pueden calcular usando regla de la cadena (en dos variables).

$$\begin{cases} y^{(0)}(t) &= f^{[0]}(t,y) = f(t,y), \\ y^{(k+1)}(t) &= f^{[k+1]}(t,y) = f_t^{[k]}(t,y) + f_y^{[k]}(t,y)\,f(t,y), \quad k = 0,1,2,...,m. \end{cases}$$

Entonces, sacando h a factor,

$$y(t_{i+1}) \approx y_{i+1} = y_i + h\left[f^{[0]}(t_i,y_i) + \frac{h}{2}f^{[1]}(t_i,y_i) + +\frac{h^2}{3!}f^{[2]}(t_i,y_i) + ... + \frac{h^{m-1}}{m!}f^{[m-1]}(t_i,y_i)\right], \; i = 0,1,...,n. \tag{6.4}$$

Ejemplo 6.2

Consideremos el problema de valor inicial $\dfrac{dy}{dt} = 0.7y - t^2 + 1, \;\; t \in [1,2], \quad y(1) = 1$. La solución exacta es $y(t) = 1.42857t^2 + 4.08163t - 4.42583e^{0.7t} + 4.40233$.

Vamos a aplicar el método de Taylor de orden $m = 4$ con $n = 10$. Tenemos $a = 1,\; b = 2,\; h = 0.1$ y $t_i = 1 + 0.1i$. Ahora debemos calcular las derivadas,

Ejemplo 6.2 (continuación).

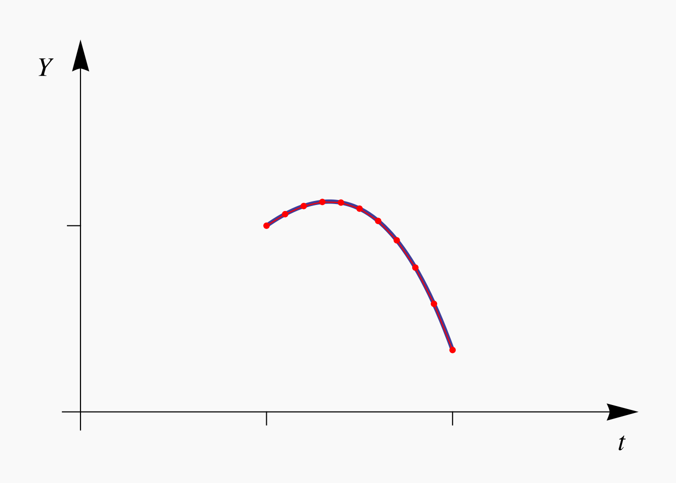
$$\begin{aligned} f^{[0]}(t,y) &= 0.7y - t^2 + 1, \\ f^{[1]}(t,y) &= -2t + 0.7(0.7y - t^2 + 1), \\ f^{[2]}(t,y) &= -2 - 2\cdot 0.7^1 t + 0.7^2(0.7y - t^2 + 1), \\ f^{[3]}(t,y) &= -2\cdot 0.7^1 - 2\cdot 0.7^2 t + 0.7^3(0.7y - t^2 + 1). \end{aligned}$$

$$\begin{cases} y_0 = 1 \\ y_{i+1} = y_i + h\left[f^{[0]}(t_i,y_i) + \frac{h}{2} f^{[1]}(t_i,y_i) + \frac{h^2}{3!} f^{[2]}(t_i,y_i) + \frac{h^3}{4!} f^{[3]}(t_i,y_i)\right],\ i = 0,1,...,n. \end{cases}$$

Así, $y_1 = y_0 + 0.1\left[f^{[0]}(t_0,y_0) + \frac{0.1}{2} f^{[1]}(t_0,y_0) + \frac{0.1^2}{3!} f^{[2]}(t_0,y_0) + \frac{0.1^3}{4!} f^{[3]}(t_0,y_0)\right] = 1.06193158375.$

En la tabla se continúa el cálculo hasta y_{10}.

t_i	Método de Taylor de orden 4 y_i	Valor exacto $y(t_i)$
1.0	1.0	1.0
1.1	1.06193158375	1.061931432036279
1.2	1.105577521330614	1.105577223160223
1.3	1.127540292137498	1.127539827069448
1.4	1.124176027574661	1.124175372973675
1.5	1.091576648813122	1.091575779638891
1.6	1.025550709391196	1.025549597968614
1.7	0.92160284874683	0.92160146451555
1.8	0.77491175596324	0.77491006520478
1.9	0.58030653570613	0.58030450124654
2.0	0.33224136049833	0.33223894138437



6.4 Algoritmo e implementación con wxMaxima.

Para el algoritmo vamos a reescribir la relación 6.4 como $y_{i+1} = yi + T^{[m]}(t_i,y_i)$ donde,

$$T^{[m]}(t,y) = h f^{[0]}(t_i,y_i) + \frac{h^2}{2} f^{[2]}(t_i,y_i) + ... + \frac{h^m}{m!} f^{[m-1]}(t_i,y_i).$$

En el algoritmo se usan las variables t_0, y_0 y y_1. En cada ciclo t_0 y y_0 se actualizan para calcular la siguiente aproximación $y_1 = T^{[}k](t_0, y_0)$. Luego se incrementa t_0 y se actualiza $y_0 = y_1$.

Algoritmo 6.2: Método de Taylor de orden m

Datos: $f(t,y)$, m, a, b, y_0, n
Salida: Imprime las aproximaciones (t_i, y_i), $i = 0,1,...,n$

$h = (b-a)/n$;
$t_0 = a$;
for $i = 1$ **to** n **do**
    $y_1 = y_0 + T^{[m]}(t_0, y_0)$;
    $t_0 = a + i \cdot h$;
    $y_0 = y_1$;
    print((t_0,y_0));

Implementación en wxMaxima. En esta primera implementación, definimos la función $f(t,y)$ y luego la llamamos en el programa por su nombre `'f'`. Por supuesto, la derivadas parciales se calculan con `fk:f(t,y)` y `fk:diff(fk,t) + diff(fk,y)*f(t,y)`,

```
f(t,y):=0.7*y-t^2+1; /*dy/dt = f(t,y)*/
ODETaylor(f,orden,a,b,y00,n):=block(
[h:(b-a)/n*1.0, t0, y0:y00,y1,fk,Tk],
t0:a,
Tk:h*f(t,y),
fk:f(t,y),
/*Polinomio T^[k](t,y)*/
for k:2 thru orden do (
          fk:diff(fk,t) + diff(fk,y)*f(t,y),
          Tk:Tk + h^k/factorial(k)*fk
                   ),
/*Cálculo de y_{i+1}*/
for i:1 thru n do (
       y1:y0+ev(Tk,t=t0,y=y0),
       t0:a+i*h,
       y0:y1,
       print("{",t0,",", y0,"}")
                   )
);
f(t,y):=0.7*y-t^2+1;
ODETaylor(f,4,1,2,1,10);
```

```
/* Salida */
1.1  ------  1.06193158375
1.2  ------  1.105577521330614
1.3  ------  1.127540292137498
1.4  ------  1.124176027574661
1.5  ------  1.091576648813122
1.6  ------  1.025550709391196
1.7  ------  0.92160284874683
1.8  ------  0.77491175596324
1.9  ------  0.58030653570613
2.0  ------  0.33224136049833
```

Para agregar el gráfico en el programa, para propósitos didácticos, debemos agregar cada nuevo punto `(t0,y0)` en una matriz. Aquí calculamos la solución exacta `yt` de la ecuación diferencial (si hubiera) y usamos `implicit(yt,t,a,b,y,-3,3)` para graficar. Se imprime los valores exactos $y(t_i)$. En vez de `-3,3` se podría usar $[\text{mín}\{yi\}-1, \text{máx}\{yi\}+1]$.

```
/*Programa con propósitos didácticos*/
/*Se asume que la solución de la EDO es expl\'{\i}cita y=y(t)*/
f(t,y):=0.7*y-t^2+1; /*dy/dt = f(t,y)*/
ODETaylor(f,orden,a,b,y00,n):=block(
[h:(b-a)/n*1.0, t0, y0:y00,y1,fk,Tk,tiyi],

t0:a,
tiyi:[[t0,y0]],
Tk:h*f(t,y),
fk:f(t,y),
/*Polinomio de T^[k](t,y)*/
for k:2 thru orden do (
   fk:diff(fk,t) + diff(fk,y)*f(t,y),
   Tk:Tk + h^k/factorial(k)*fk
                      ),
/*Cálculo de y_{i+1}*/
for i:1 thru n do (
        y1:y0+ev(Tk,t=t0,y=y0),
        t0:a+i*h,
        y0:y1,
        tiyi:append(tiyi,[[t0,y0]]),
        print(tiyi[i+1] ," --- ", ev(yt,t=t0,y=y0))
                    ),

/*Solución y(t)*/
        eq:'diff(y,t) = f(t,y),
        sol:ode2(eq, y, t),
        yt:ic1(sol,t= a,y= y00),
        print("------    ",yt),
/*gráfico*/
        wxdraw2d(
        implicit(yt,t,a,b,y,-3,3),
        color=red, point_size=1, point_type=filled_circle,points(tiyi),
        yrange=[-4,4], xrange=[0,5]
            )
);

f(t,y):=0.7*y-t^2+1;
ODETaylor(f,4,1,3,1,10);
```

6.5 Métodos de Runge-Kutta.

Como decíamos, los métodos de un solo paso tienen la forma

$$y_{i+1} = y_i + h \cdot \Phi(t_i, y_i; h), \quad h > 0.$$

En el método de Euler la *función incremento* es

$$\Phi(t_i, y_i; h) = f(t_i, y_i)$$

Para el método de Taylor de orden 2 es,

$$\Phi(t_i, y_i; h) = f(t_i, y_i) + \frac{h}{2}\left[f_t(t_i, y_i) + f_y(t_i, y_i) f(t_i, y_i)\right] \tag{6.5}$$

Los métodos de Runge-Kutta son métodos diseñados pensando en imitar las expansiones de Taylor pero usando solo evaluaciones de la función $f(t, y)$. En el caso del método de Runge-Kutta de orden 2, se trata de modificar el método de Euler escribiendo

$$\Phi(t_i, y_i; h) = a_1 k_1 + a_2 k_2, \tag{6.6}$$

con $k_1 = f(t, y)$ y $k_2 = f(t + \alpha h,\, y + \beta h k_1)$, es decir, no se va a evaluar en la tangente hasta $t_i + h$ sino antes, usando la pendiente de la tangente en $(t + \alpha h,\, y + \beta h k_1)$.

Expandiendo 6.5 y 6.6 en potencias de h (usando la fórmula de Taylor en dos variables) y comparando se obtiene, entre varias opciones, $\alpha = 1$, $\beta = 1$, $a_1 = a_2 = 1/2$. Esto nos da un método Runge-Kutta de orden 2,

$$y_{i+1} = y_i + \frac{h}{2}\left[f(t_i, y_i) + f(t_i + h,\, y_i + h f(t_i, y_i))\right]$$

El método clásico de Runge-Kutta de orden 4 tiene una función de incremento que coincide con el polinomio de Taylor hasta el sumando con el término h^4. Este método se puede escribir como,

$$\begin{cases} y_0 &= y(a), \\ y_{i+1} &= y_i + \frac{1}{3}(k_1 + 2k_2 + 2k_3 + k_4),\ i = 0, 1, 2, \ldots \end{cases}$$

donde

$$\begin{aligned} k_1 &= \frac{h}{2} f(t_i, y_i), \\ k_2 &= \frac{h}{2} f\left(t_i + \frac{h}{2},\, y_i + k_1\right), \\ k_3 &= \frac{h}{2} f\left(t_i + \frac{h}{2},\, y_i + k_2\right), \\ k_4 &= \frac{h}{2} f\left(t_i + h,\, y_i + 2k_3\right), \end{aligned}$$

6.6 Algoritmo e implementación con WXMAXIMA.

Algoritmo 6.3: Método de Runge-Kutta de orden 4

Datos: $f(t,y)$, m, a, b, y_0, n

Salida: Imprime las aproximaciones (t_i, y_i), $i = 0,1,...,n$

$h = (b-a)/n;$

$t_0 = a;$

for $i = 1$ **to** n **do**

$\quad k_1 = \frac{h}{2} f(t_i, y_i);$

$\quad k_2 = \frac{h}{2} f\left(t_i + \frac{h}{2}, y_i + k_1\right);$

$\quad k_3 = \frac{h}{2} f\left(t_i + \frac{h}{2}, y_i + k_2\right);$

$\quad k_4 = \frac{h}{2} f(t_i + h, y_i + 2k_3);$

$\quad y_1 = y_i + \frac{1}{3}(k_1 + 2k_2 + 2k_3 + k_4);$

$\quad t_0 = a + i \cdot h;$

$\quad y_0 = y_1;$

$\quad$ print$((t_0, y_0))$;

Implementación en WXMAXIMA.

```
f(t,y):=0.7*y-t^2+1; /* dy/dt = f(t,y) */
RungeKutta(f,a,b,y00,n):= block(
[h:(b-a)/n*1.0, t0: a,  y0:y00, y1, k1,k2,k3,k4],

print(t_i, "  ----  ", y_i, "    y(t_i)      "),
print(t0, "  ----  ", y0, " ----- ", y0),
/* Cálculo de los y_{i+1}*/
...
);

f(t,y):= t^2-y^2;
RungeKutta(f,0,2,1,10);
```

EJERCICIOS

6.1 Considere el problema de valor inicial $y' = \cos(2t) + \text{sen}(3t)$, $t \in [0,1]$, $y(0) = 1$.

a) Usando el método de Euler, aproximar $y(0.4)$ con $h = 0.1$.

b) Usando el método de Taylor de orden 4, aproximar $y(0.4)$ con $h = 0.2$

c) Usando el método de Runge-Kutta de orden 4, aproximar $y(0.4)$ con $h = 0.2$

6.2 Considere el problema de valor inicial $y' = t\,\mathrm{Exp}(3t) - 40y$, $t \in [1,2]$, $y(1) = 10$.

a) Usando el método de Euler, aproximar $y(0.4)$ con $h = 0.1$.

b) Usando el método de Taylor de orden 4, aproximar $y(0.4)$ con $h = 0.2$

c) Usando el método de Runge-Kutta de orden 4, aproximar $y(0.4)$ con $h = 0.2$

6.3 Usando el método de Runge-Kutta de orden 4, aproximar $y(0.2)$ (con $h = 0.1$) si $y(t) = \int_0^t e^{-t^2}\,dt$. (Debe convertir el cálculo de la integral en un problema de valor inicial.)

6.7 Algunos Detalles Teóricos.

Definición 6.1

Consideremos un conjunto $D \subseteq \mathbb{R}^2$ y una función $f(t,y)$ definida en D. Si existe una constante $L > 0$ tal que

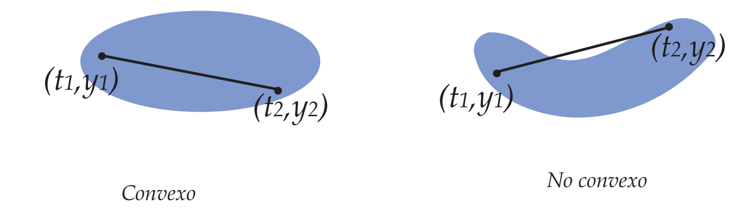
$$|f(t,y_1) - f(t,y_2)| \leq L|y_1 - y_2|, \quad \forall (t,y_1), (t,y_2) \in D$$

se dice que $f(t,y)$ cumple una condición de Lipschitz en la variable y en D. A L se le llama constante de Lipschitz para f.

Nota: Una condición *suficiente* para que $f(t,y)$ cumpla una condición de Lipschitz en D es que exista $L > 0$ tal que $\left|\frac{\partial f}{\partial y}(t,y)\right| \leq L, \quad \forall (t,y) \in D$

Definición 6.2

Un conjunto $D \subseteq \mathbb{R}$ se dice convexo si $\forall (t_1,y_1), (t_2,y_2) \in D$, el segmento $\{(1-\lambda)(t_1,y_1) + \lambda(t_2,y_2), \quad \lambda \in [0,1]\}$ está contenido en D.



Nota: Observe que, cuando $\lambda = 0$ estamos en el punto inicial (t_1,y_1) y cuando $\lambda = 1$ estamos en el punto final (t_2,y_2). $\lambda = 1/2$ corresponde al punto medio del segmento.

Teorema 6.2

Si $D = \{(t,y) : a \leq t \leq b, -\infty \leq y \leq \infty\}$ y si $f(t,y)$ es continua en D y satisface una condición de Lipschitz respecto a y en D, entonces el problema $(*)$ tiene una solución única $y(t)$ para $a \leq t \leq b$.

Nota: Un problema de valor inicial está *bien planteado* si pequeños cambios o perturbaciones en el planteo del problema (debido a errores de redondeo en el problema inicial, por ejemplo), ocasiona cambios pequeños en la solución del problema. Si un problema cumple las hipótesis del teorema anterior, entonces está bien planteado.

Ejemplo 6.3

Consideremos el problema de valor inicial $y' = y - t^2 + 1$, $t \in [0,4]$, $y(0) = 0.5$. Aquí $f(t,y) = y - t^2 + 1$. Si $D = \{(t,y) : 0 \leq t \leq 4, -\infty \leq y \leq \infty\}$, entonces como

$$\left|\frac{\partial f}{\partial y}(t,y)\right| = |1| \quad \forall (t,y) \in D$$

f cumple una condición de Lipschitz en y (en este caso podemos tomar $L = 1$). Además, como $f(t,y)$ es continua en D, el problema de valor inicial tiene una solución única. De hecho la única solución es $y(t) = (t+1)^2 - 0.5e^t$

6.8 Estimación del error

Teorema 6.3

Si $D = \{(t,y) : a \leq t \leq b, -\infty \leq y \leq \infty\}$ y si $f(t,y)$ es continua en D y satisface una condición de Lipschitz respecto a y en D con constante L entonces si existe una constante M tal que

$$|y''(t)| \leq M, \ \forall t \in [a,b]$$

entonces para cada $i = 0,1,2,...,n$,

$$|y(t_i) - y_i| \leq \frac{hM}{2L}\left(e^{L(t_i - a)} - 1\right)$$

Nota: para calcular $|y''(t)|$ usamos regla de la cadena: $y''(t) = \dfrac{\partial f}{\partial t}(t,y) + \dfrac{\partial f}{\partial y}(t,y) \cdot y'(t)$. Posiblemente sea difícil obtener M dado que puede ser necesaria información acerca de $y(t)$.

Teorema 6.4 (Teorema de Taylor).

Sea $h > 0$ y f una función tal que f y sus primeras k derivadas son continuas en el intervalo $[a, a+h]$ y la derivada f^{k+1} existe en $]a, a+h[$, entonces existe un número ξ , en el intervalo $]a, a+h[$ tal que

$$f(a+h) = f(a) + \frac{f'(a)}{1!}h + \frac{f''(a)}{2!}h^2 + \cdots + \frac{f^{(k)}(a)}{k!}h^k + \frac{f^{(k+1)}(\xi)}{(k+1)!}h^{k+1}$$



Apéndice A

Programación con LibreOffice Basic (=OOoBasic).

En este apéndice nos ocupamos de los constructores básicos del lenguaje OpenOffice.org Basic (OOo Basic) o de LibreOffice Calc, que es básicamente lo mismo. Luego consideramos las funciones y las subrutinas con el propósito de empezar a construir *una biblioteca* con funciones y subrutinas de uso frecuente en la implementación de algoritmos en métodos numéricos.

OpenOffice.org 3.x (`http://es.openoffice.org/`) es una suite ofimática (procesador de textos, hoja de cálculo, presentaciones, etc.) libre, disponible para varias plataformas, tales como Microsoft Windows, GNU/Linux, BSD, Solaris y Mac OS X.



Figura A.1 Inicio de OpenOffice.org

*

En lo que nos concierne, vamos a usar el lenguaje de programación OOo Basic y la hoja electrónica OOo Calc. OOo Basic es una abreviación de "OpenOffice.org Basic". Calc es una hoja de cálculo si-milar a Excel y la programación de macros es muy similar a VBA para Excel. No se puede decir, en realidad, si es más sencillo usar VBA u OOo Basic; ambos hacen las cosas sencillas, pero a su manera.

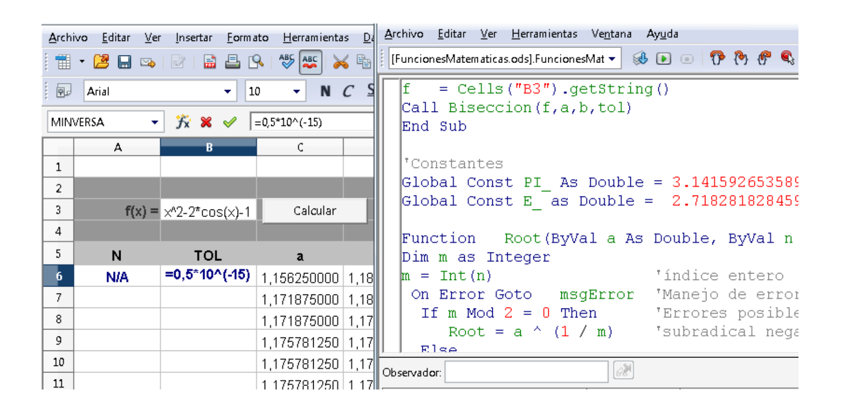


Figura A.2 OOo Calc

VBA y OOo Basic son lenguajes de programación de la familia "Basic". Como tal, comparten los mismos constructores linguísticos básicos, por ejemplo, declaración de variables, ciclos (For, Do, While, Do While, Do Until,...), condicionales, operadores lógicos, funciones, etc. La diferencia está en el API, es decir, la manera de comunicarse con los documentos de OpenOffice.org. y el modelo de objetos de cada uno. Por tanto, si alguien ya está familiarizado con algún lenguaje de de la familia Basic (por ejemplo, VBA), se sentirá cómodo con OOo Basic.

Este capítulo está orientado a estudiar las construcciones necesarias para programar algoritmos en matemática e ingeniería, que se puedan usar en conjunto con OOo Calc. La programación de este capítulo está orientada a construir una biblioteca con subrutinas y funciones de propósito general y de uso frecuente en la programación de algoritmos en métodos numéricos y otras ramas.

A.1 Preliminares: Macros, funciones y subrutinas.

Una "macro" es una una colección ordenada de instrucciones o comandos. Las macros son programas, están constituidas por constantes, variables, instrucciones, subrutinas y funciones. Las subrutinas y funciones son programas y son un caso particular de macros.

A.1.1 Editar y ejecutar una macro.

Lo primero que hay que aprender (y recordar) es que las macros se guardan en *módulos*, estos a su vez se guardan y organizan en *bibliotecas*, las cuales, están contenidas dentro de documentos (cuadernos OOoCalc). Hay una biblioteca default llamada "`Standard`". Por ahora vamos a poner nuestras macros en un módulo en esta biblioteca `Standard`.

Sesión de programación en OOoCalc, con OOo Basic. Los pasos generales para hacer nuestro primer programa son: Preparamos la hoja OOoCalc, abrimos el entorno de programación, implementamos el programa (macro) y luego ejecutamos (y depurar si es el caso).

Nuestro primer programa es muy sencillo: Una macro que despliega una ventana con el mensaje "Hola!".



Figura A.3 Primer programa: Una macro que despliega una ventana con el mensaje "Hola!".

Pasos para implementar y ejecutar el programa.

a) Abrimos un libro OOoCalc y los guardamos, digamos como "LeerImprimir". Así, tendremos un archivo `LeerImprimir.ods` (el cual puede abrir en windows, Linux o Mac!).

b) Ahora vamos a crear un módulo llamado "`Module1`" (su nombre default) para editar subrutinas y funciones. Para esto vamos a

`Herramientas>Macros>Organizar macros>OpenOffice.org Basic`

y en la ventana `Macros básicas`, (1) seleccionamos nuestra hoja `LeerImprimir`, (2) hacemos clic en '`Nuevo`' y (3) hacemos clic en `Aceptar`. Esto nos llevará al "Entorno de Desarrollo Integrado" (IDE por sus siglas en inglés) de OOo Basic. La manera directa de hacer esto es presionar `Alt-F11-[Nuevo o Editar]`).

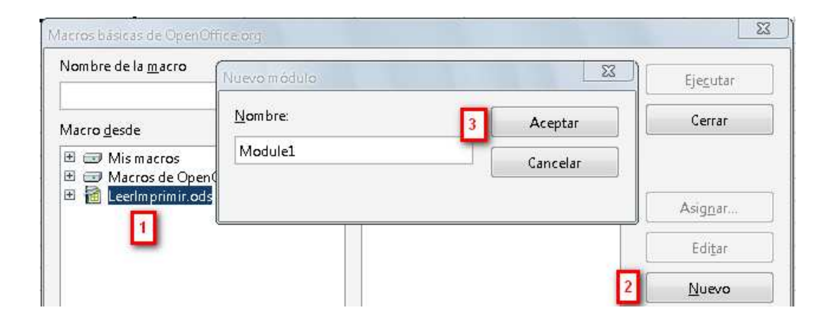


Figura A.4 Crear Módulo

c) En el IDE de OOo Basic aparece la subrutina default "`Main`". Nuestro primer programa es muy simple: Enviamos un mensaje `Hola!` usando el comando `MsgBox`. Cuando entramos al IDE encontramos una `sub`rutina vacía

```
Sub  Main

 End Sub
```

Agregamos el código con el mensaje (el apóstrofo "'" se usa para poner comentarios),

Programa 1 *Utilizando la subrutina `Main` para levantar una ventana con un mensaje (figura A.5).*

```
Sub  Main
'MsgBox "Mensaje", #=tipo de ventana, "título de la ventana"
  MsgBox "Hola!" , 64,"Primer programa"
End Sub
```

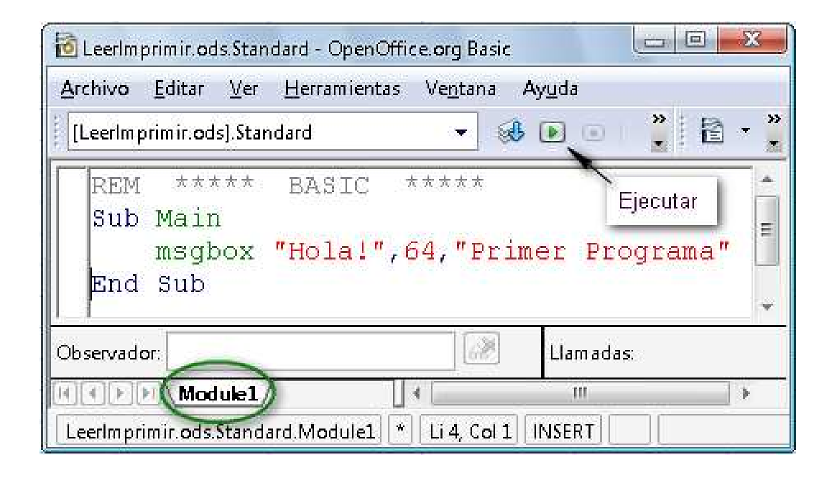


Figura A.5 IDE de OOo Basic. Subrutina Main con un mensaje

d) Para *ejecutar* la subrutina `Main` podemos usar el botón de ejecución (ver figura A.5) o, desde el cuaderno, usar la combinación de teclas `Alt-F11` seleccionar y presionar el botón de `Ejecutar`.

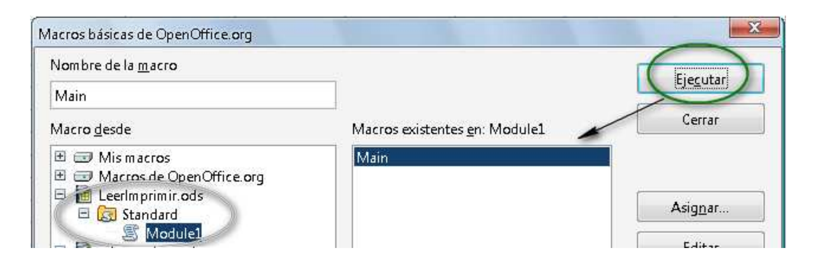


Figura A.6 Ejecutar la subrutina con `Ctrl-F11`

Bibliotecas y módulos. Los módulos se deben poner en bi-bliotecas. La biblioteca default es la biblioteca `Standard`. En la figura A.6 se observe que el `módulo 1` quedó en esta biblioteca. Más adelante volveremos sobre este tema.

Edición Los comentarios inician con `REM` o con un apóstrofo recto (`'`). El compilador no es sensitivo a las mayúsculas y minúsculas, es decir, es lo mismo escribir `MaxIteraciones` que `maxIteraciones`.

Ventanas de mensaje. Hay varios tipos de ventanas de mensaje. En el siguiente código se muestra cuatro opciones.

Programa 2 *Tipos de ventanas de mensaje*

```
Sub  Main
 MsgBox " ", 16, "Icono  Stop"          'Mensaje vacío
 MsgBox " ", 32, "Icono  pregunta"
```

```
 MsgBox " ", 48, "Icono  exclamación"
 MsgBox " ", 64, "Icono  información"
End Sub
```



Figura A.7 Iconos para MsgBox (mensaje vacío)

A.1.2 Subrutinas y funciones.

Una subrutina es una colección de instrucciones (es un caso especial de macro). Podemos usar una subrutina para incluir las instrucciones para leer una valor de una celda e imprimir en otra celda. Una función es una macro que devuelve (generalmente) un valor. Podemos usar una función para implementar las funciones usuales en matemática.

Cuando ejecutamos una subrutina, se ejecutan las instrucciones que contiene. En OOo Basic hay una subrutina default, la subrutina `Main`.

```
Sub  Main

End Sub
```

Esta es la primera subrutina que se ejecuta. En principio, como no tiene instrucciones, no pasa nada.

La estructura de una función es (lo que está entre "[ ]" es opcional),

```
Function NombreFuncion([argumentos])
    Variables
    instrucciones
  [ Exit  Function]
    instrucciones
  NombreFuncion= lo que retorna la función
End Function
```

La línea opcional `Exit Function` se usa en el caso de que, por alguna razón, queramos detener el cálculo y salir de la función

La estructura de una subrutina es (lo que está entre "[ ]" es opcional),

```
Sub  NombreSubrutina([argumentos])
    Variables
    instrucciones
    [ Exit Sub]
    instrucciones
End Sub
```

La línea opcional `Exit Sub` se usa en el caso de que, por alguna razón, queramos terminar ahí la subrutina.

Funciones matemáticas. Las funciones matemáticas se implementan de manera natural. Considere $f(x) = x^3 + x + 1$ y $g(x,y) = \sqrt{x^2 + y^2}$. La función f tiene un argumento y la función g tiene dos argumentos.

Programa 3 *Implemetación de las funciones $f(x) = x^3 + x + 1$ y $g(x,y) = \sqrt{x^2 + y^2}$.*

```
1 Function   F(x)
2       F = x^2+x+1
3 End Function
4
5 Function   G(x,y)
6      G = Sqr(x^2+y^2)
7 End Function
```

En la línea `6.` se usa la función Basic `Sqr` para la raíz cuadrada.

Usar una función en una hoja. Una vez definida la función en un módulo de la biblioteca `Standard`, ya es accesible en el cuaderno actual y se pueden usar en las fórmulas. Solo *un detalle*: En el cuaderno los argumentos se separan con punto y coma (“;”) mientras que en el ambiente OOo Basic se usa la coma corriente.

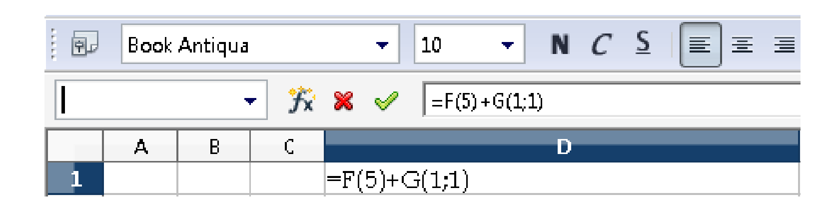


Figura A.8 Usando una funciones en el cuaderno OOoCalc

A.1.3 Variables.

Las variables contienen valores que pueden cambiar en la ejecución de una macro. Aunque OOo Basic no nos obliga a declarar variables, en la práctica es mejor hacerlo.

Los nombre de las variables inician con una letra (A−Z o a−z). Se pueden usar números (0−9) y el guión bajo (_) pero no al principio.

“Option Explicit”. Es bueno declarar las variables y también es muy práctico agregar la instrucción "`Option Explicit`" para detectar nombres que no corresponden a alguna variable. Esto va a ahorrar mucho tiempo a la hora de buscar

errores en nuestro código.

Dim. Para declarar una variable se usa la palabra reservada “`Dim`” y se puede agregar el tipo `Integer` para enteros y `Long` para enteros grandes, `Double` para números en doble precisión, `String` para cadenas de caracteres, `Boolean` para variables que toman los valores verdadero o false, etc.

Variables tipo `Variant`. Sino se declara el tipo, se asume que es tipo “`Variant`” y todo se acomoda al tipo de dato. No siempre es bueno usar el default “`Variant`” porque a veces no es claro, después de algunas asignaciones, en que tipo de dato se va a convertir este “`Variant`”.

Variable tipo `Integer`. Las variables tipo `Integer` se inicializan en 0 y soportan valores entre -32768 y 32767.

Variable tipo `Long`. Las variables tipo `Long` se inicializan en 0 y soportan valores entre -2147483648 y 2147483647.

Variable tipo `Double`. Las variables tipo `Double` se inicializan en $0{,}0$ y soportan números positivos y negativos entre $\pm 1{,}79769313486232 \times 10^{308}$ y $\pm 4{,}94065645841247 \times 10^{-324}$.

Los computadores comunes representan los números reales en este rango con un número aproximado. El error *relativo* entre el número real x y su aproximación $\tilde{x}$ en el computador, es menor o igual que 0.5×10^{-15}. Los números positivos inferiores a $4.94065645841247 \times 10^{-324}$ se tratan como 0.

El computador representa los números `Double` con a lo sumo 15 o 16 dígitos significativos.

		Número real	*Representación en el computador*
c_1	=	299792458112345699	2,99792458112346E+017
$c_2 = c_1 + 16$	=	299792458112345715	2,99792458112346E+017
20!	=	243290200817664	2,43290200817664E+018
21!	=	5109094217170944	5,10909421717094E+019

Los números c_1, c_2 tienen más de 15 dígitos significativos, en la suma real difieren pero la suma en el computador resulta igual. Hay que tomar en cuenta que en una aproximación, 16 es porcentualmente despreciable respecto al número 299792458112345699. Por otra parte, 20! se puede representar de manera exacta (como un `Double`) pero 21! (16 dígitos) se representa aproximadamente.

Como decíamos más arriba, si la representación del número real x es $\tilde{x}$, entonces

$$\left|\frac{x-\tilde{x}}{x}\right| \leq 0.5 \times 10^{-15}$$

Programa 4 *Declaración de variables y algunos cálculos.*

```
Option Explicit
Sub  Prueba()
Dim a,b               'a y b son de tipo Variant
Dim q1    As Integer  'q es de tipo entero
Dim q2    As Double   'q es de tipo Double

```

```
7  'Las variables se pueden inicializar en una
8  'sola línea si las  separamos con ":"
9   a   = 5665555737832145 : b = a+20
10 MsgBox b                 '-> 5,66555573783216E+015
11  q1  = a/b               'División real
12 MsgBox q1                '-> 1
13  q2  = a\b                 'División Entera: -> Error
14 End Sub
```

El cálculo en la línea `11.`, `q1=a/b` devuelve 1 pues por la precisión que maneja el computador (15 o 16 dígitos), `a=b+20`.

El cálculo en la línea `13.`, `q2=a\b` provoca un error en tiempo de corrida: La división entera espera dividir dos enteros y retornar un entero, si `a` y `b` no son enteros, deben ser truncados para convertirlos en enteros, pero en el ejemplo estos números están fuera del rango de los enteros. Se despliega una venta de error con el mensaje `"Tipo de datos o valor inadmisible. Desbordamiento."`

Cadenas de caracteres (String). En el manejo de texto es necesario conocer algunas operaciones sobre cadenas de caracteres ((String).) Los espacios *en blanco* cuentan como caracteres.

- Declaración de variables: `Dimtxt As String`
- Pasar un número x a String: `Str(x)`
- Concatenar: `txt = Str(x)& " es real"` o también `Str(x) + " es real"`.
- Accesar partes de un String:
  (a) `Left(txt, n)` muestra los primeros `n` caracteres de `txt`.
  (b) `Right(txt, n)` devuelve los últimos `n` caracteres de `txt`.
  (c) `Mid (txt, m, n)` devuelve los primeros `n` caracteres de `txt` desde la posición `m`.
  (d) `Len(txt)` devuelve el número de caracteres de `txt`.
  (e) `Trim(txt)` elimina espacios iniciales y finales. `Trim(" Hola! ") -> "Hola!"`.

Programa 5 *Ejemplo para mostrar el uso de algunos métodos en cadenas de caracteres.*

```
Sub  Texto()
Dim txt     As String
Dim rt      As String
Dim lg      As Integer

 txt = "Texto de prueba"
 rt   =  Left(txt,6)     'rt -> "Texto ", con un blanco al final.
 rt   =  Right(txt, 5)   'rt -> "rueba", sin blancos.
 rt   =  Mid(txt, 8, 5)  'rt -> "e pru".
 lg   =  Len(txt)        'lg -> 15.

'Mid con cuatro argumentos pasa a ser  instrucción, es decir,
'modifica la tira txt pero no devuelve algo.
```

```
'La siguiente instrucción modifica txt, cambia " de " por "=".
 Mid(txt, 6, 4, "=")          'Ahora txt = "Texto=prueba"

End Sub
```

- Buscar y reemplazar: La instrucción `InStr(PosInicial, txt, txtBuscado)` devuelve una número con la posición en la que se encontró la primera aparición, desde la posición `PosInicial`, de la subcadena `txtBuscado` en la cadena `txt`. Este comando ignora mayúsculas y minúsculas. También se puede usar `Mid` con cuatro argumentos como se muestra en el ejemplo,

Programa 6 *Uso de* `InStr`.

```
Sub  Posicion()
Dim txt      As String
Dim i      As Integer

 txt = "Texto de prueba"
 i =  inStr(1,txt," de ")    '-> i=6, pues " de " inicia en un blanco.

 Mid(txt, 6, 4, "=")          'Ahora txt = "Texto=prueba"
 MsgBox  inStr(1,txt," de ") '-> i=0
End Sub
```

Ámbito de las variables. Las variables *locales* son las que se declararan dentro del cuerpo de una subrutina o función y se crean al invocar ésta y se destruyen al finalizar. Si estas variables se declaran `Static` (en vez de `Dim`) entonces conservan el último valor que tuvieron, entre llamada y llamada (siempre y cuando no cambie la macro que llama a la macro que la contiene). En el siguiente código se muestra la misma función pero con los dos tipos de variable.

```
Sub  Main
 MsgBox Dsucesor(1)  'Imprime 1
 MsgBox Dsucesor(1)  'Imprime 1
 MsgBox Dsucesor(1)  'Imprime 1
                     '---------
 MsgBox Stsucesor(1) 'Imprime 1
 MsgBox Stsucesor(1) 'Imprime 2
 MsgBox Stsucesor(1) 'Imprime 3
End Sub

Function  Dsucesor(n  As Integer)  As Integer
Dim el_sucesor  As Integer
 el_sucesor = el_sucesor+n
  Dsucesor = el_sucesor
End Function

Function  Stsucesor(n  As Integer)  As Integer
 Static el_sucesor  As Integer
```

```
 el_sucesor = el_sucesor+n
  Stsucesor = el_sucesor
End Function
```

Las variables *globales de dominio público* de un módulo se declaran al inicio del módulo y son visibles para todos los módulos de la biblioteca. Se pueden usar y modificar en las subrutinas y funciones. No conserva su último valor.

```
 Option Explicit
Dim ValorGeneral  As Integer

Sub Main
End Sub
```

Las variables *globales* son las mismas que las variables de dominio público excepto que mantienen su último valor aunque se termine la macro que la utilizó. Se declaran al inicio del módulo y son visibles para todos los módulos del Cuaderno Calc. Se pueden usar y modificar en las subrutinas y funciones.

```
 Option Explicit
 Global ValorG  As Integer 'ValG es global y conserva su último valor

Sub Main
End Sub
```

A.1.4 Constantes

Para declarar una constante, usamos la palabra clave `Const` de la siguiente manera,

```
 Const PI  As Double = 3.14159265358979
 Const E   As Double =  2.71828182845905
```

Para que las constantes sean visibles para todos los módulos usamos `Global` (por ser constantes no se pueden modificar).

```
'PI_ y E_ son visibles para todos los módulos de este cuaderno
 Global Const PI  As Double = 3.14159265358979
 Global Const E  As Double =  2.71828182845905
```

Constantes para OOoCalc. Las constantes definidas en OOo Basic se pueden usar en la hoja OOo Calc si se implementan como una función, digamos en la biblioteca `Standard`. π ya tiene una implementación, la base e de los

logaritmos naturales se implementa así,

Programa 7 *Constante e y función* `E_()`

```
 Global Const E As Double =  2.71828182845905
                        'π para la hoja: Ya está definida como PI()
Function E_()          'Constante e para la hoja. La función debe tener
   E_=2.71828182845905 'un nombre diferente al de la constante.
End Function
```

Ahora podemos evaluar en la hoja fórmulas con esta función, por ejemplo `"=2*E_()^2"`.

Nota. En OOo Basic se usa `E` para escribir números en notación científica, pero esto no presenta problemas con la constante `E`. Por ejemplo OOo Basic entiende `0.1E+2` como `10` y entiende `0.1E+2+E` como `12.71828182...`

A.1.5 Operadores

Como es usual, las operaciones aritméticas son `+`, `-`, `*` y `^` para los exponentes. El operador `"+"` también se usa para concatenar cadenas de caracteres. En las tablas (**??**) y (**??**) se muestra una lista de operadores y funciones de uso frecuente.

Operador	*Significado*	*Ejemplo*
`Mod`	Resto de la división entera.	`-5 mod 3 -> -2`
`\`	División entera	`2\3 =0` y `7\4 =1.`
`&`	Concatenación de cadenas	`"xi =" & "g(xi)" -> "xi=g(xi)"`
`<=`	$\leq$	
`>=`	$\geq$	
`<>`	$\neq$	`4 mod 3 <> 0 -> true`
`Not`	Operador lógico $\neg$ (negación).	
`AND`	Operador lógico `"y"`	
`OR`	Operador lógico `"o"`	
`XOR`	Operador lógico `"ó"`	`True XOR True = False, True XOR False = True`
`ABS(x)`	El valor absoluto de un número	`Abs(-2) -> 2`
`CLng(x)`	Redondea al `Long` más cercano.	
`Fix(x)`	Trucar la parte decimal.	
`Int(x)`	Parte entera (entero de la izquierda).	`Int(3.99)= 3, Int(-0.47)= -1.`
`CInt(x)`	Redondea al entero más cercano.	`CInt(3.99)= 4, CInt(-0.47)= 0`
`Str(x)`	Convierte en `String`	`Str(2.45) ->"2.45"`
`CDbl(x)`	Convierte a `Double`	`CDbl("3,2")+3 -> 6,2;  CDbl("0,45E3") -> 450`
`Val(x)`	Convierte en `Double`	`Val("3.2") +3 -> 6,2;  Val("0.45E2") -> 450`
`Rnd()`	número aleatorio entre 0 y 1	`2*rnd() - 1 -> número aleatorio en [-1,1]`
`SGN(x)`	signo	`sgn(-5.5) -> -1, sgn(0)-> 0`
`SQR(x)`	$\sqrt{x}$	
`Log(x)`	El logaritmo natural de un número.	
`Exp(x)`	e^x	
`SIN(x)`	$\operatorname{sen}(x)$	
`COS(x)`	$\cos(x)$	
`TAN(x)`	$\tan(x)$	
`ATN(x)`	Arcotangente	`atan(x)` $\in]-\pi/2, \pi/2[$

Notas. Según las reglas de conversión implícitas, `"3,2"+3` pasa a ser la cadena `"3,23"` mientras que `CDbl("3,2")+3` pasa a ser el número `6,2`. Hay que tener en cuenta que `CDbl` respeta la configuración regional de idioma y reconoce la coma como separador decimal, esto significa que en español ignora el punto: `CDbl("3.2")+3 ->35`. En cambio `Val` reconoce el punto como se-parador decimal pero devuelve el resultado de acuerdo a la configuración regional, por ejemplo `Val("3.2")+3 ->6,2`; `Val("0.1E2")-> 10` pero `Val("0,1E2")-> 100`.

A.1.6 Ciclos.

En métodos numéricos es muy frecuente acumular sumas o productos y manejar esquemas iterativos que requieren repetir algunos cálculos hasta que se cumpla alguna condición, esto se hace con estructuras de código que manejan ciclos.

Ciclo For. Este ciclo repite un bloque de instrucciones un número determinado de veces. La sintaxis general es (lo que está entre "[ ]" es opcional),

```
For  contador = inicio To fin [ Step ValordeSalto ]
      instrucciones...
      [ Exit For ]
Next [contador]
```

Acumular sumandos. Vamos a implementar una función `ElSumatorio(ini,n)` para calcular el sumatorio

$$\sum_{i=\text{ini}}^{n} 1/2^i.$$

Este ejemplo muestra una idea muy usada para acumular sumandos. Para calcular el sumatorio $\sum_{i=\text{ini}}^{n} 1/2^i$ procedemos de manera natural. Usamos una variable `suma` con valor inicial cero y acumulamos cada nuevo sumando, uno por uno,

$$\begin{array}{llllllll}
 & \texttt{suma} & = 0, & & & & & \\
i=0, & \texttt{suma} & = & \texttt{suma} & +1/2^0 & = & 0+1 & = & 1, \\
i=1, & \texttt{suma} & = & \texttt{suma} & +1/2^1 & = & 1+1/2 & = & 3/2, \\
i=2, & \texttt{suma} & = & \texttt{suma} & +1/2^2 & = & 3/2+1/4 & = & 7/4, \\
\ldots & & & & & & & &
\end{array}$$

Programa 8 *Un ejemplo de cómo acumular sumandos: Un sumatorio*

```
Function ElSumatorio(ini, n)
Dim i, suma
    suma=0
   For  i= ini To n
        suma = suma + 1/2^i
    Next i
    ElSumatorio = suma
End Function
```

Acumular factores. Para mostrar la manera de acumular factores en un producto vamos a implementar la función `factorial(n)`. Recordemos que $n! = 2\cdot 3\cdots(n-1)\cdot n$.

Procedemos de manera natural. Usamos una variable `producto` con valor inicial 1, acumulamos cada nuevo factor, uno por uno,

$$\begin{array}{llllll} & \texttt{producto} & = 1, & & & \\ i=2, & \texttt{producto} & = & \texttt{producto*2} = 1*2 & = & 2 \\ i=3, & \texttt{producto} & = & \texttt{producto*3} = 2*3 & = & 6 \\ i=4, & \texttt{producto} & = & \texttt{producto*4} = 6*4 & = & 24 \\ \ldots & & & & & \end{array}$$

Programa 9 *Un ejemplo de cómo acumular factores: La función factorial.*

```
Function Factorial(n)'1≤ n ≤ 170. Valores exactos hasta n=20.
Dim i, producto
    producto=1
   For  i= 2 To n
        producto = producto*i
    Next i
    Factorial = producto
End Function
```

Podemos usar la misma idea del programa anterior para implementar una función `CoefBinomial(s,n)`. Recordemos que si $s \in \mathbb{R}$ y $n \in \mathbb{N}$, $\binom{s}{0} = 1$ y $\binom{s}{1} = s$ y en general,

$$\binom{s}{n} = \frac{s(s-1)(s-2)\ldots(s-n+1)}{n!}.$$

Procedemos de manera natural. Usamos una variable `producto` con valor inicial 1, acumulamos cada nuevo factor, uno por uno. El código es,

Programa 10 *Coeficiente binomial* $\binom{s}{n}$.

```
Function CoefBinomial(s,n)
Dim i, producto

    producto=1
    If n>0 Then
     For  i= 0 To n-1
            producto = producto*(s-i)/(i+1)
      Next i
    End If
    CoefBinomial = producto
End Function
```

Ciclo Do...Loop. Este ciclo viene en diferentes sabores. Se utiliza para hacer la ejecución de un bloque de código mientras o hasta que una condición se cumpla. El uso más común es verificar la veracidad de la condición antes de ejecutar el código. El código se ejecuta repetidamente mientras la condición sea `true`. Si la condición es falsa, el código nunca se ejecuta. La sintaxis general es (lo que está entre "[ ]" es opcional),

```
Do While condición
  instrucciones...
  [ Exit Do]
  instrucciones...
Loop
```

En otra forma del ciclo `Do...Loop`, el código se ejecuta repetidamente mientras la condición es falsa. En otros palabras, el código se ejecuta hasta que la condición se convierte en verdad. *Si la condición se evalúa como verdadera al inicio, el ciclo nunca se ejecuta.* La sintaxis general es (lo que está entre "[ ]" es opcional),

```
Do Until condición
  instrucciones...
 [ Exit Do]
  instrucciones...
Loop
```

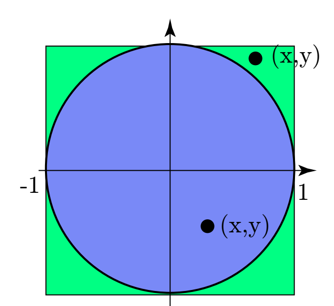
Se puede colocar el control al final del ciclo, en cuyo caso *el bloque de código se ejecuta al menos una vez*. En este caso, el bucle se ejecuta al menos una vez y luego se ejecuta varias veces mientras la condición sea verdadera. La sintaxis general es (lo que está entre "[ ]" es opcional),

```
Do
  instrucciones...
  [ Exit Do]
  instrucciones...
Loop While condición
```

Para ejecutar el ciclo al menos una vez y luego continuar mientras la condición sea falsa, utilice el siguiente constructor (lo que está entre "[ ]" es opcional),

```
Do
  instrucciones...
  [ Exit Do]
  instrucciones...
Loop Until condición
```

Ciclo While ... Wend. Este ciclo se usa cuando se quiere repetir un bloque de código mientras una condición es `true`. Este ciclo no ofrece beneficios adicionales al ciclo `Do...Loop`, por ejemplo no hay un "`Exit While`". La sintaxis general es



Números aleatorios en un círculo. Para obtener pares (x,y) aleatoriamente distribuidos en un círculo de radio 1 se generan dos números aleatorios x, y entre -1 y 1. Este par estará en un cuadrado de lado 2 centrado en el origen. Este círculo tiene área π mientras que el cuadrado tiene área 4. Si generamos un par (x,y) aleatorio con $-1 < x,y < 1$, la probabilidad de que este par quede dentro del círculo es $\pi/4$. Los pares (x,y) en el círculo de radio 1 tienen la propiedad $x^2 + y^2 \leq 1$.
Como `rnd()` genera un número aleatorio entre 0 y 1, entonces $-1 \leq$ `2*rnd()-1` ≤ 1. El par aleatorio lo generamos con `x = 2*rnd()-1` y `y = 2*rnd()-1`. Esto lo hacemos *mientras que* $\sqrt{x^2+y^2} > 1$ (el programa que sigue usa arreglos, una exposición más amplia de los arreglo se puede ver en A.5.5).

Programa 11 *Generar pares (x,y) aleatoriamente distribuidos en un círculo de radio 1*

```
Sub  Main
  Dim p()
    p =  xyCAleatorio()'generamos el par aleatorio en el círculo
  MsgBox  Str(p(0))+"  "+Str(p(1))
End Sub

Function  xyCAleatorio()
Dim x,y
Dim par()
  par = Array(0,0) 'inicializamos el par
Do 'ciclo se ejecuta de nuevo solo si (x,y) queda fuera del círculo
  x = 2*rnd()-1
  y = 2*rnd()-1
Loop While Sqr(x^2+y^2)>1
 par(0)= x 'p(0) es la componente x
 par(1)= y 'p(1) es la componente y
  xyCAleatorio=par()
End Function
```

El número esperado de veces que se ejecuta el ciclo `Do` es $4/\pi \approx 1.27$ veces!

Números armónicos. Los números armónicos son los números $H_N = \sum_{n=1}^{N} 1/n$. En análisis matemático se establece que $H_N \approx \ln(N) + 0.57721$ para N grande (ver [32]). El siguiente programa calcula el $N-$ésimo número armónico usando un ciclo `Do...Loop Until`,

Programa 12 *Números armónicos $H_N = \sum_{i=1}^{N} 1/i$*

```
Function NHarmonico(N)
Dim i As Long
Dim suma As Double
    suma=0 : i=1
    Do
      suma = suma+ 1/i
         i = i+1
    Loop Until  i > N
    NHarmonico = suma
End Function
```

=NHARMONICO(I2)

I	J	K
N	H_N	Ln(N) +0,57721
10	2,9289682539682500	2,8797950929940500
100	5,1873775176396200	5,1823801859880900
10000	9,7876060360443500	9,7875503719761800
100000	12,0901461298633000	12,0901354649702000
1000000	14,3927267228650000	14,3927205579643000

Figura A.9 Comparando H_N con $\ln(N) + 0.57721$

A.1.7 Condicionales.

La condición `If` es usado para ejecutar un bloque de código de acuerdo a si se cumple o no una condición. La forma más sencilla de esta instrucción es

```
If condición Then
        instrucciones...
End If
```

La condición puede ser cualquier expresión que se evalúa con `true` o `false`. La sintaxis general es (lo que está entre "[ ]" es opcional),

```
If condición1 Then
        instrucciones...
[Else If condición2 Then]
        instrucciones...
[Else]
        instrucciones...
End If
```

Si la primera condición que se evalúa es `true`, se ejecuta el primer bloque de código. Se pueden usar varias declaraciones del tipo `ElseIf` para probar otras condiciones. La declaración `Else` se ejecuta si ninguna de las otras condiciones se evalúa como `true`.

Una función a trozos. Vamos a implementar la función

$$T(a,x) = \begin{cases} a\ln(x) & \text{si} \quad x > 0, \\ \cos(a/x) + \text{sen}(a/x) + e^x & \text{si} \quad x < 0, \\ 0 & \text{si} \quad x = 0. \end{cases}$$

No podemos usar `T(a,x)` pues `T` ya está reservado para una función que trabaja con texto. Para no tener problemas, cambiamos el nombre a `fT`. El código podría ser,

Programa 13 *Función* `T(a,x)`

```
Function  fT(a,x)
 Dim Tax
 If x>0 Then
         Tax=Log(x)
 Else If x<0 Then
         Tax=cos(a/x)+sin(a/x)+Exp(x)
 Else
         Tax=0
 End If
      fT= Tax
End Function
```

A.2 Leer e imprimir en una celda.

En lo que nos concierne, lo primero que tenemos que aprender es cómo leer *datos numéricos* de las celdas y como imprimir datos numéricos en una o varias celdas. Para leer datos de una celda se debe indicar la hoja que vamos a usar, para esto usamos el objeto `ThisComponent` que designa el documento desde donde vamos a llamar la macro que editamos. La hoja 1 corresponde a la hoja 0.

Leer o imprimir en la celda usando el nombre. Para leer o imprimir el valor numérico almacenado en una celda usando el nombre de la celda, se usa el método `getCellRangeByName` y la propiedad `Value`.

Programa 14 *Leer x en* `"A4"` *e imprimir $f(x)$ en* `"B4"`*, desde la subrutina* `Main`.

```
Sub  Main
Dim Hoja, x
 Hoja = ThisComponent.Sheets(0)            'Hoja 0
 x    = Hoja.getCellRangeByName("A4").Value'Lee el valor numérico en "A4"
                                           'y lo almacena en x.
 Hoja.getCellRangeByName("B4").Value = f(x)'Imprime f(x) en "B4"
End Sub

```

```
Function   F(x)
      F = x^2+x+1
End Function
```

	A	B	C
1			
2			
3	x	f(x)	
4	4	21	

Figura A.10 Leer $x = 4$ en "A4" e imprimir $f(4) = 21$ en "B4"

Leer o imprimir en la celda usando la posición. La hoja electrónica se puede ver como una matriz.

A1 '(0,0)

	A	B	C
1	(0,0)	(1,0)	(2,0)
2	(0,1)	(1,1)	(2,1)
3	(0,2)	(1,2)	(2,2)

Figura A.11 La celda A1 corresponde celda $(0,0)$.

La celda (c, f) corresponde a la columna c y la fila f. La columna A es la columna 0, la columna B es la columna 1, etc. La celda "A1" es la entrada $(0,0)$, la celda "A2" es la celda $(0,1)$,la celda "A3" es la celda $(0,2)$ y la celda "An" es la celda $(0, n-1)$. La celda "Bn" es la celda $(1, n-1)$, etc.

Para leer o imprimir un *valor numérico* almacenado en una celda usando su posición en la matriz, se usa el método getCellByPosition y la propiedad Value.

Programa 15 *Leer x en* "A4"=(0,3) *e imprimir $f(x)$ en* "B4"=(1,3)*, desde la subrutina* Main.

```
Option Explicit
Sub Main
Dim Hoja, x
  Hoja = ThisComponent.Sheets(0)          'Corresponde a la Hoja 1
  x    = Hoja.getCellByPosition(0,3).Value'Contenido numérico de "A4"
  Hoja.getCellByPosition(1,3).Value = f(x)'imprime f(x) en "B4"
End Sub
```

Podemos ahora imprimir una comparación entre los números armónicos y la aproximación $H_N \approx \ln(N) + 0.57721$. En el cuaderno de la figura (A.12) los números armónicos se imprimen en las celdas J2,J3,J4,... que corresponde a las celdas (9,i) con $i = 1,2,...$ y el valor de comparación se imprime en las celdas K1,K2,... que corresponde a las celdas (10,i) con $i = 1,2,...$

	J	K
1	H_N	Ln(N) +0,57721
2	2,9289682539682500	2,8797950929940500
3	5,1873775176396200	5,1823801859880900
4	7,4854708605503400	7,4849652789821400
5	9,7876060360443500	9,7875503719761800
6	12,0901461298633000	12,0901354649702000

Figura A.12 Comparando H_N con $\ln(N) + 0.57721$

Programa 16 *Imprimir Números armónicos* $H_N = \sum_{i=1}^{N} 1/i$ *vs* $\ln(N) + 0.57721$

```
Sub Main()
 Dim i
 For i=1 To 5
  ThisComponent.Sheets(0).getCellByPosition(9,i).Value = NHarmonico(10^i)
  ThisComponent.Sheets(0).getCellByPosition(10,i).Value = Log(10^i)+0.57721
 Next i
End Sub


Function NHarmonico(N)
Dim i As Long
Dim suma As Double
    suma=0 : i=1
    Do
      suma = suma+ 1/i
         i = i+1
    Loop Until  i > N
    NHarmonico = suma
End Function
```

Para leer o imprimir *texto* se usa el método `setString`, por ejemplo

Programa 17 *Leer e imprimir texto en una celda*

```
Sub Main
Dim txt
  txt= "Si x= "+Str(x)+",  f(x)= "+Str(f(x))
  ThisComponent.Sheets(0).getCellByPosition(1,3).setString(txt)
End Sub
```

Leer cadenas de caracteres de una celda Para leer el contenido de una celda como una cadena de caracteres (String) se usa el método `getString`, por ejemplo

	A	B
1		
2		
3	x	f(x)
4	4	Si x= 4, f(x)= 21

Figura A.13 Imprimir texto

```
Dim txt  As String
 txt   = ThisComponent.Sheets(0).getCellRangeByName("A4").getString()
```

Color. De paso, para cambiar el color de fondo y el color de la fuente en la celda usamos `CharColor` y `CellBackColor`,

```
Dim oCell
 oCell = ThisComponent.Sheets(0).getCellRangeByName("A4")
 oCell.Value = 2.343403000102
'Color del texto: blanco
 oCell.CharColor = RGB(255,255,255)
'Color de fondo: azul
 oCell.CellBackColor = RGB(0,0,255)
```

A.3 Ejecutar una subrutina (o una función) desde un botón.

Para dar un ejemplo de cómo se inserta un botón en la hoja y de cómo se le asigna una subrutina (de tal manera que cuando el usuario hace clic en el botón, se ejecuta la subrutina), vamos a implementar una subrutina de nombre "`LeerImprimir()`". Agregamos las instrucciones en la subrutina para leer un valor x_0 en la celda "`A4`" e imprimir $f(x_0)$ en la celda "`B4`". La función f será $f(x) = x^3 + x + 1$.

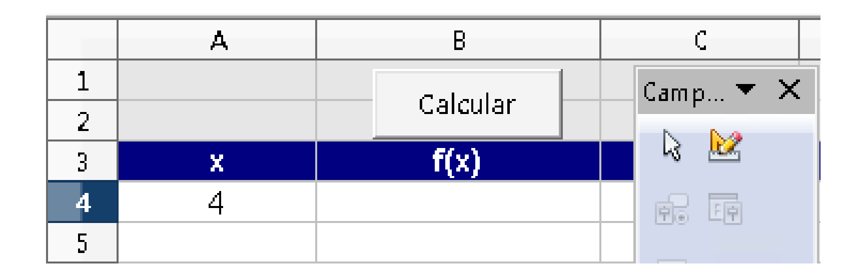


Figura A.14 Leer e imprimir en una celda

El código de la subrutina podría ser

Programa 18 *Subrutina para leer e imprimir en una celda*

```
Sub LeerImprimir()
   Dim Hoja,x
```

```
    Hoja = thisComponent.Sheets(0)
    x    = Hoja.getCellRangeByName("A4").Value
    Hoja.getCellRangeByName("B4").Value = f(x)
End Sub
```

Después de digitar el código procedemos a la *compilación* para revisar la sintaxis. La sintaxis, en el ambiente computacional, se refiere al "conjunto de reglas que definen las secuencias correctas de los elementos de un lenguaje de programación". Estas reglas son las que estamos aprendiendo en el camino. Observe que la sintaxis no tiene que ver con la lógica del cálculo. Lo que tratamos de hacer es que la sintaxis esté bien y el cálculo sea correcto.

Agregar el botón. Ahora sigue insertar el botón que "dispara" el cálculo. Debemos tener visible la barra "`Campos de control de formulario`". Si no esta visible, se habilita con

`Ver>Barra de herramientas>Campos de control de formulario.`

Para crear un botón, seleccionamos el botón en la barra `Campos de control` y arrastramos el ratón (presionando el botón izquierdo) en cualquier parte del cuaderno.

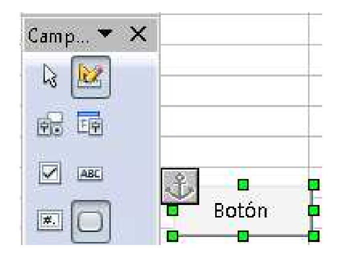


Figura A.15 Selecciona el botón y arrastra el ratón...

En este momento, el botón está en "modo diseño". Para poner la etiqueta "Calcular" abrimos la ventana de propiedades del botón (con clic derecho sobre el botón, abrimos el menú "`Campo de control`")

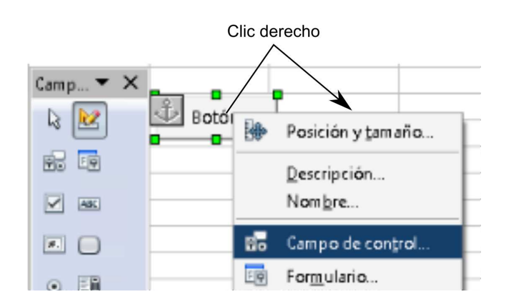


Figura A.16 Campos de control.

y editamos el campo "`Título`".
Ahora, seleccionamos la cejilla "`Acontecimientos`" y asignamos una acción: "`Al iniciar`" (cuando el usuario hace clic en el botón). Elegimos la macro, es decir, la función o subrutina que se va a ejecutar cuando el usuario hace clic en el botón. En nuestro caso, seleccionamos la subrutina `LeerImprimir()`.
Salir de modo diseño. Luego de hacer clic en "`Aceptar`" y cerrar la ventana "`Propiedades: Botón`", debemos habilitar el botón (que hasta ahora ha estado en "modo diseño"): hacemos clic en (en la barra "`Campos de control de formulario`"). Ahora ya podemos hacer clic en el botón para realizar el cálculo e imprimir. Una subrutina se puede

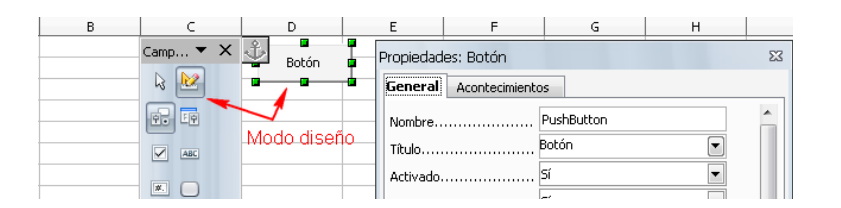


Figura A.17 El botón está en modo diseño, podemos agregar propiedades.

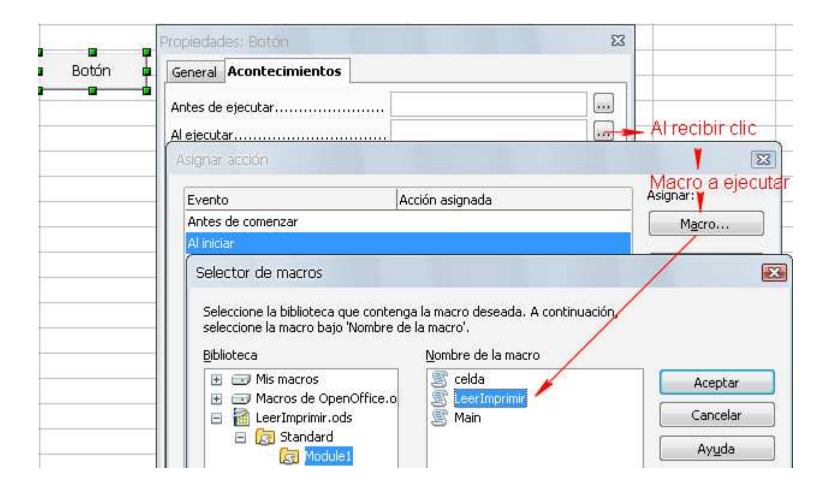


Figura A.18 Asignar la subrutina que se ejecuta cuando se hace clic en el botón.

ejecutar también con `ALT-F11`.

A.4 Crear, exportar, importar y cargar bibliotecas.

Recordemos de nuevo que las macros se guardan en módulos, estos a su vez se guardan y organizan en bibliotecas, las cuales están contenidas dentro de documentos. Hasta ahora, todos nuestras subrutinas y funciones las hemos editado en un módulo en la biblioteca `Standard`. La biblioteca `Standard` *no* se puede exportar (ni eliminar), así que si queremos tener disponibles nuestras funciones especiales para otros cuadernos, una opción es exportarlas en otra biblioteca. Así otro cuaderno las puede importar.

Para ver el proceso de exportar e importar, vamos a reunir varias funciones matemáticas y otras utilitarias e introducirlas en un módulo de una nueva biblioteca. Esta nueva biblioteca la llamaremos aquí `BblMatematica`. Luego *exportamos* esta biblioteca para que cualquier otro cuaderno la pueda *importar*. Los pasos son,

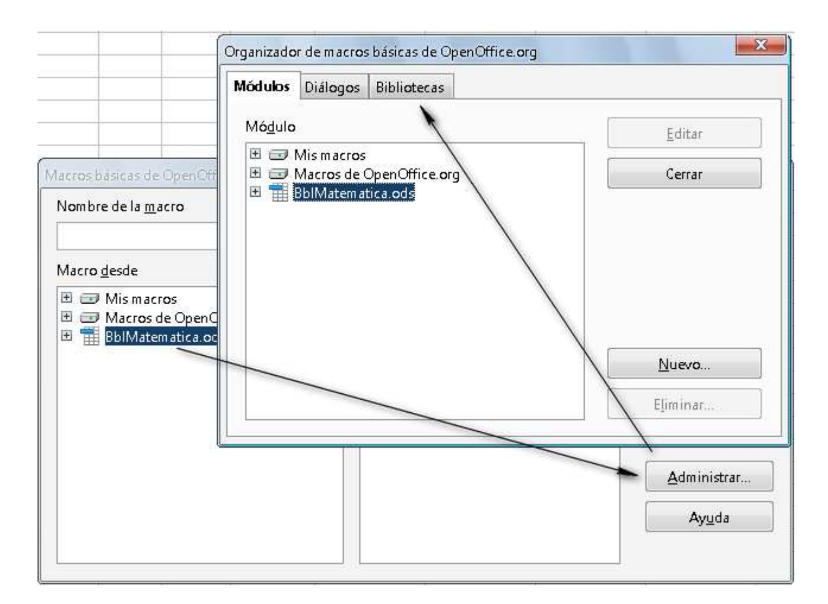
1. Crear una nueva biblioteca `BblMatematica`
2. Exportar la biblioteca
3. Importar y *cargar* la biblioteca (en otro cuaderno)

A.4.1 Crear una biblioteca.

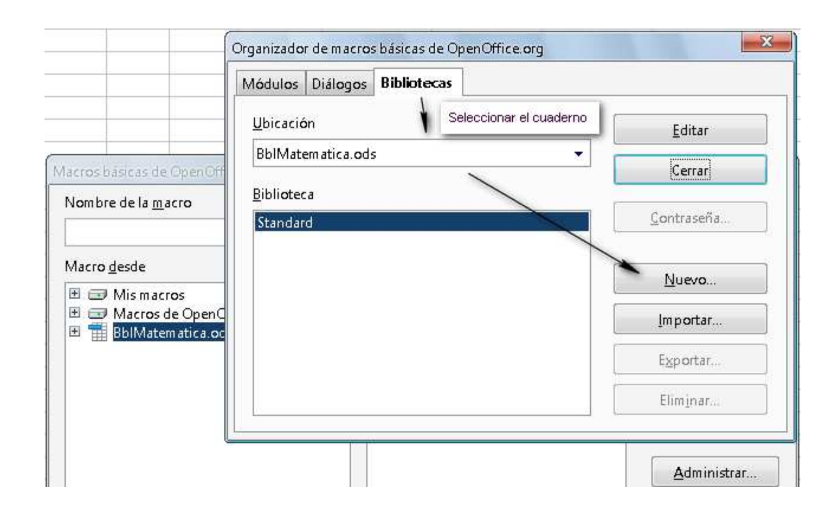
En lo que sigue, vamos a suponer que tenemos un cuaderno llamado `BblMatematica`. En este cuaderno tenemos las funciones y las macros especiales que queremos exportar como una biblioteca. Vamos a suponer que todos estos

programas están en la biblioteca `Standard`.

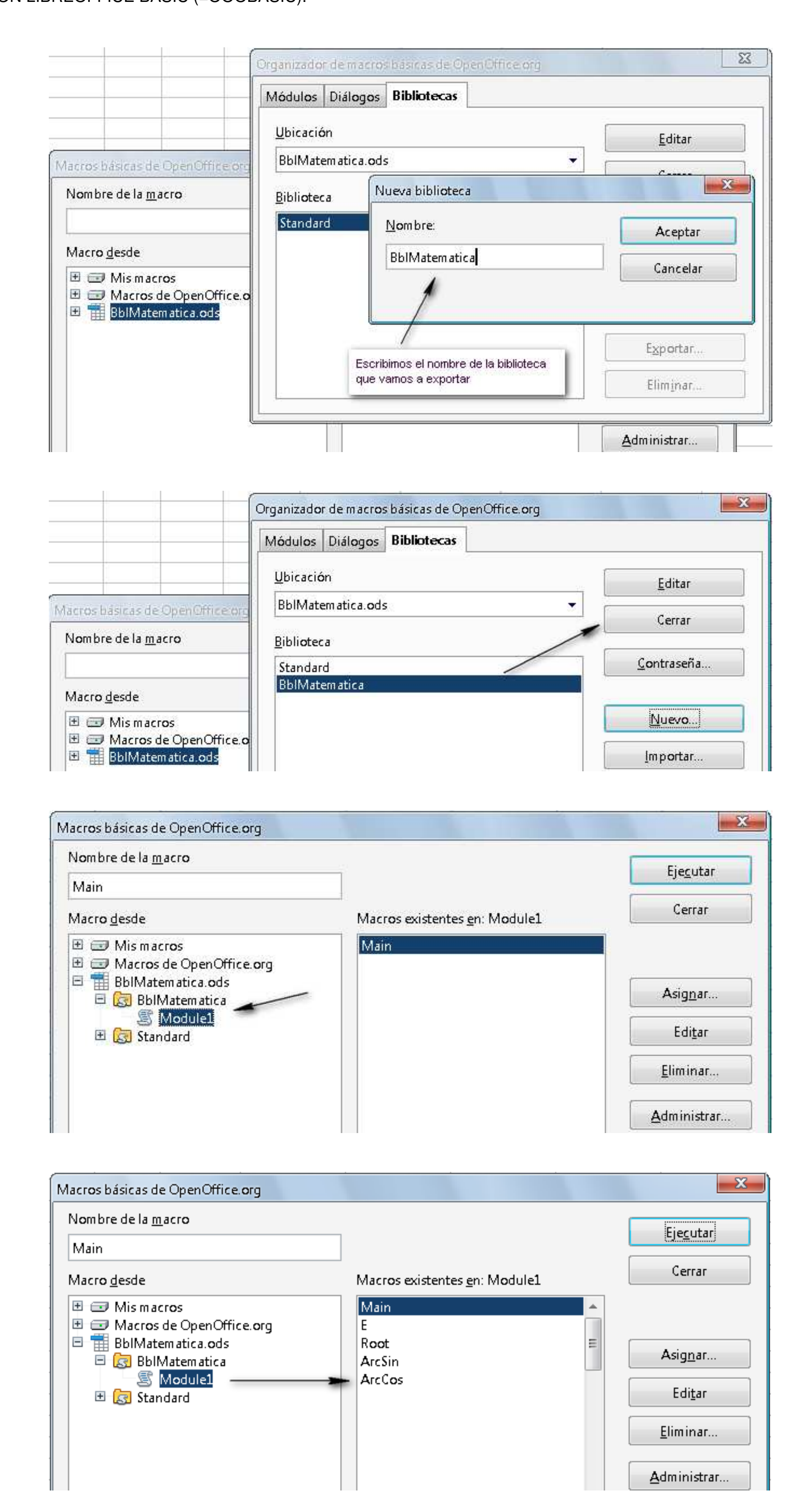
Para crear esta biblioteca (que contendrá el módulo o los módulos con las funciones especiales u otras macros) abrimos la ventana `Macros básicas` con `ALT-F11`. Hacemos clic en el botón `Administrar`.
En la nueva ventana `Organizador de macros...` elegimos la pestaña `Bibliotecas` (`Library`),



en la cejilla `Ubicación`(`Location`) elegimos el cuaderno actual,

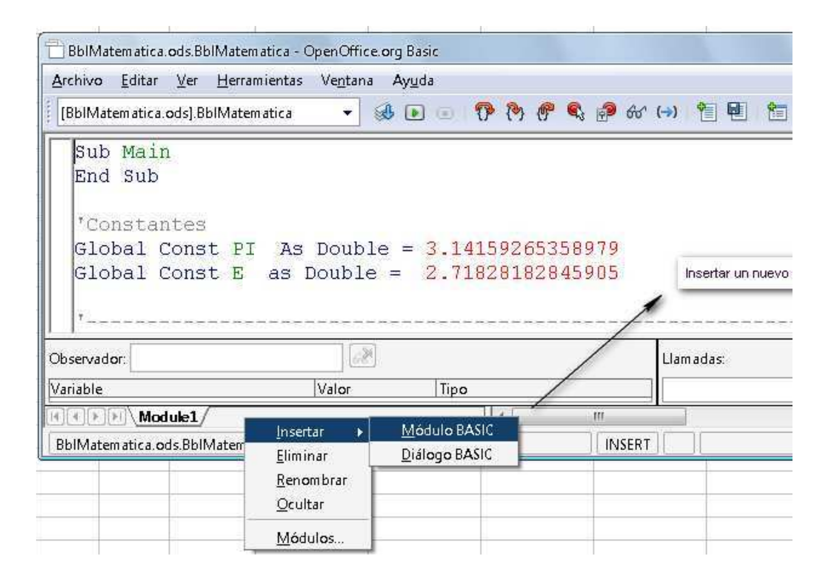


y presionamos el botón `Nuevo`. Luego, en la nueva ventana ponemos el nombre a nuestra bi-blioteca, en el campo correspondiente. Puede ser cualquier nombre, en nuestro caso le pondremos el mismo nombre del cuaderno de trabajo: `BblMatematica`.
Aparece la biblioteca agregada. Podemos hacer clic en `Cerrar (Close)`.
La nueva biblioteca tiene ya un `module 1`.
Presionamos el botón `Editar` para introducir el código de algunas funciones y subrutinas en este módulo y para editar cualquier otra función nueva.



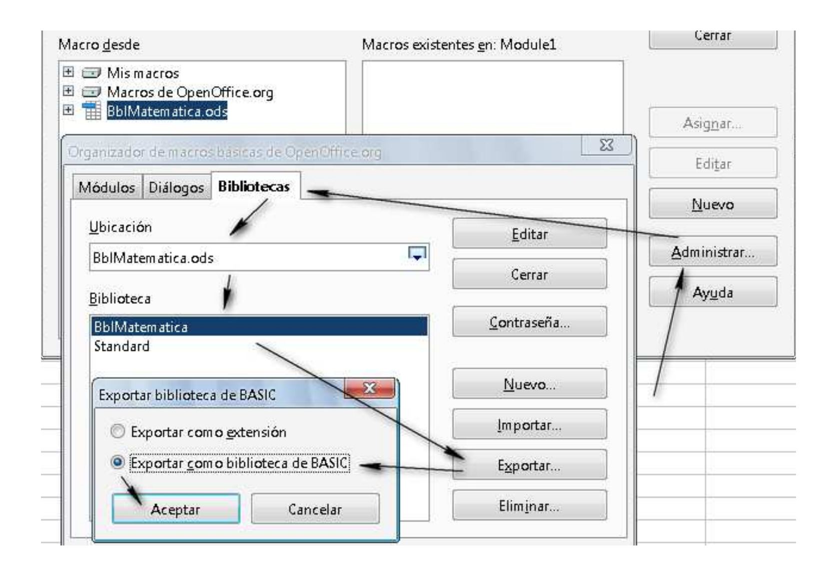
A.4.2 Agregar un nuevo módulo.

Para crear un nuevo módulo hacemos clic derecho en la barra en la que aparece el nombre del módulo y elegimos `Insertar -> Módulo Basic`.



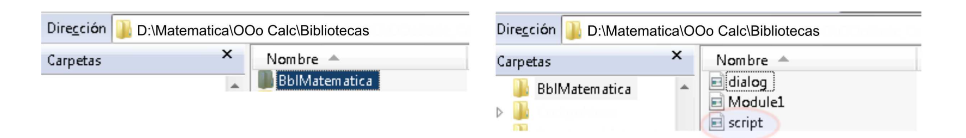
A.4.3 Exportar una biblioteca.

Para exportar una biblioteca (con nuestras macros personales u otro tipo de macro) primero abrimos el cuadro `Macros básicas...` con `ALT-F11`. Hacemos clic en el botón `Administrar`. En la nueva ventana `Organizador de macros...` elegimos la pestaña `Bibliotecas`, en la cejilla `Ubicación` elegimos el cuaderno que tiene la biblioteca y luego en la cejilla `Biblioteca` elegimos la biblioteca que vamos a exportar y presionamos el botón `Exportar`. En la nueva ventana elegimos `Exportar como biblioteca Basic` y hacemos clic en `Aceptar`. Inmediatamente se abre el explorador para ubicar la carpeta en la vamos a guardar la biblioteca.



Una vez elegida la ubicación, hacemos clic en `Aceptar` y luego hacemos clic en el botón `Cerrar`.

En nuestro caso, la biblioteca queda guardada como una carpeta con tres archivos `dialog` `Module 1` y `script`. Esta biblioteca está lista para *importar* en otro cuaderno.



A.4.4 Cargar una biblioteca

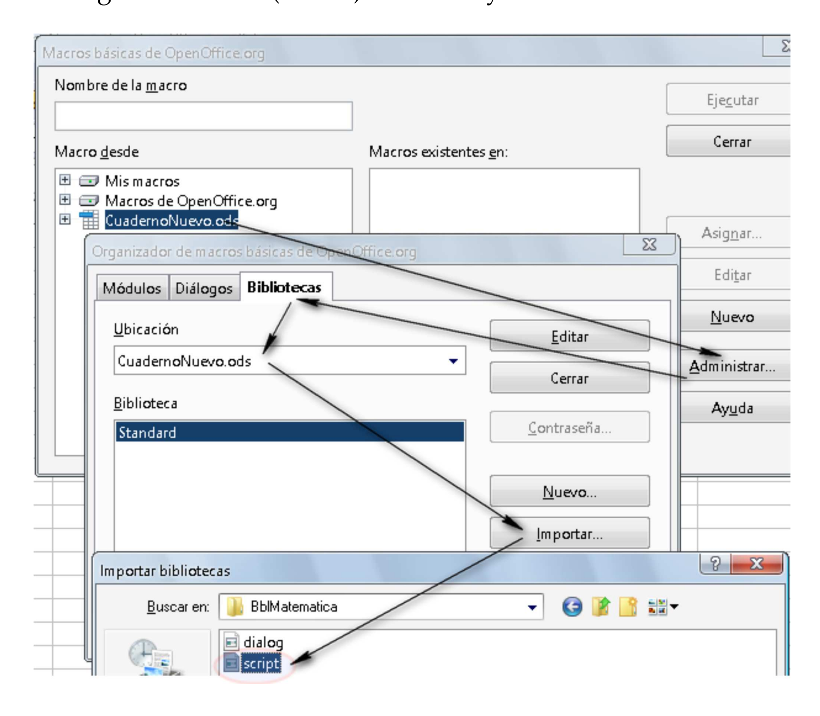
Importar una biblioteca no significa que este disponible. Cuando inicia OpenOffice.org solo se carga la biblioteca `Standard`. Este es un mecanismo para acelerar la inicialización. Para acceder a las bibliotecas importadas, se deben cargar (y el 'motor' Basic las compila). Por ejemplo, para cargar la biblioteca (ya importada) `BblMatematica` en la subrutina `Main`, se usa el código

```
'Cargar la biblioteca "BblMatematica" (si ya fue importada).
Sub  Main
  BasicLibraries.loadLibrary("BblMatematica" )
End Sub
```

Si no hacemos esto, usar alguna función de esta biblioteca provoca un error en tiempo de corrida: `Property or method not found (propiedad o método no encontrado).`

A.4.5 Importar una biblioteca.

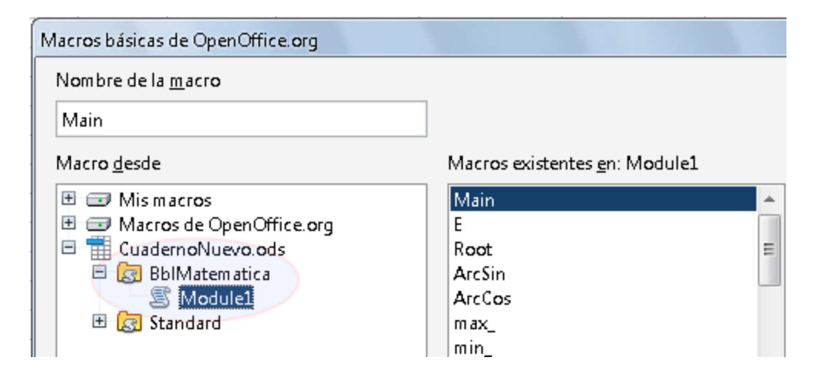
Para importar una biblioteca en un cuaderno `CuadernoNuevo`, primero abrimos el cuadro `Macros básicas...` con `ALT-F11`, presionamos el botón `Administrar`. En la nueva ventana `Organizador de macros...` elegimos la pestaña `Bibliotecas`, en `Ubicación` elegimos nuestro (nuevo) cuaderno y hacemos clic en el botón `Importar`.



Se abre el explorador y vamos a la carpeta en la que está la biblioteca, la abrimos y seleccionamos, en nuestro caso, el archivo `script.xlb`. Hacemos clic en `Abrir`.

En la nueva ventana, hacemos clic en `Aceptar` y luego cerramos la última ventana que queda.

Ahora, además de la biblioteca `Standard`, la nueva biblioteca está disponible,



Recordemos que para poder usar la biblioteca, ésta se debe cargar. En el código de la figura que sigue se usan la funciones `Cells` y `Root(x,n)` de la biblioteca `BblMatematica` (ver la sección **??**) en un módulo de la biblioteca `Standard` del nuevo cuaderno.

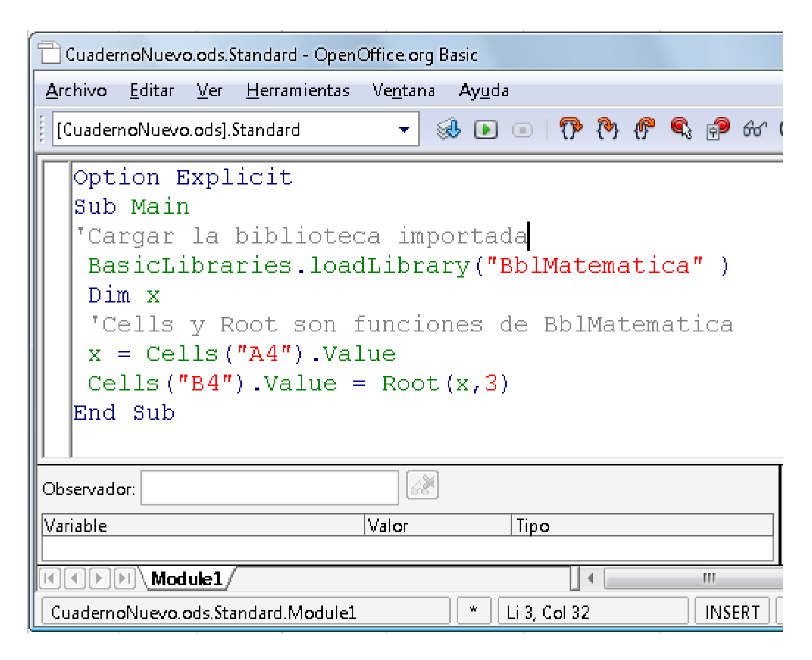


Figura A.19 Cargar y usar nuestra la biblioteca `BblMatematica`.

A.5 Subrutinas y funciones

En esta sección estudiamos aspectos más específicos de las funciones y subrutinas con el propósito de iniciar la implementación de algunas funciones especiales para una biblioteca de funciones para métodos numéricos.

A.5.1 Pasar parámetros a una subrutina o una función.

Pasar parámetros por valor y por referencia. A una subrutina o a una función se le pueden pasar los argumentos o parámetros de dos maneras, una es por valor y otra por referencia. Cuando pasamos los argumentos *por valor*, en realidad lo que se hace es pasarle una “copia” del valor de la variable, de tal manera que las modificaciones que sufra

la copia no afecta a la variable original. En cambio, cuando los argumentos se pasan *por referencia*, lo que estamos haciendo es pasarle la "ubicación" de la variable en la memoria y entonces la variable original si se puede modificar.

Por default, salvo que se indique lo contrario, *los argumentos se pasan por referencia.* Para pasarlos por valor hay que agregar `ByVal`,

```
 Function  Root( ByVal a As Double,  ByVal n As Double) As Double
...
End Function
```

Parámetros Opcionales. En las funciones y las subrutinas se pueden poner variables opcionales de cualquier tipo. Se debe comprobar si se 'paso' o no el argumento para que en su defecto,se asigne un valor por default a dicho argumento, para verificar si se paso o no un argumento se usa la función de OOo Basic `IsMissing( Argumento)`.

División con resto. Si $b \neq 0$, la división de a por b se representa de dos maneras $a = qb + r$ o como $a/b = q + r/b$ donde q es el cociente y r el resto. En el programa que sigue se hace una división entera. El parámetro opcional es `n`, si el parámetro no esta presente o si $n = 0$, el resultado se imprime como $a = qb + r$. Opcionalmente se puede escoger imprimir $a/b = q + r/b$ si agregamos el parámetro opcional con valor `n =1`.

Programa 19 *División con resto. El tipo de salida está controlada por un parámetro opcional*

```
Function  Division_yResto(a, b,  Optional n  As Integer)  As String
Dim tipoSalida As Integer

If Not IsMissing(n) Then 'Si el parámetro n está presente
  tipoSalida  = n
Else 'Si n  no está, usamos el valor default del parámetro
  tipoSalida  = 0
End If

If tipoSalida =0 Then   'Salida  b*q + r
  Division_yResto = Str(a)+"="+Str(b)+"*"+Str(a)+"+"+Str(a Mod b)
End If
If tipoSalida =1 Then    'Salida  a/b = q + r/b
  Division_yResto = Str(a)+"/"+Str(b)+"="+Str(a/b)+"+"+Str(a Mod b)+"/"+Str(b)
End If
End Function
```

Al ejecutar `Division_yResto(5,4)` devuelve `"5=4*1+1"` mientras que `Division_yResto(5,4,1)` devuelve `"5/4=1+1/4"`.

A.5.2 Manejo de errores.

Los errores en tiempo de ejecución (como divisiones por cero, desbordamiento, etc.) se pueden controlar con el manejador de errores ('ErrorHandler') de OOo Basic. Cuando el manejador de errores detecta un error, ejecuta la salida de la función (o la subrutina) en la línea de código del error e inmediatamente ejecuta el código que hemos asignado al manejador de errores. Aquí vamos a usar, en principio, un manejador de errores genérico: Cuando detectamos un

error, salimos de la función (o la subrutina) y enviamos una mensaje con la descripción del error. El código sería el siguiente,

```
Function  nombreFuncion()  'Manejador genérico de errores

 On Error Goto  msgError
   '...
   '...el código de la función va aquí
    nombreFuncion =....
 Exit  Function
'Código para 'manejar' el error
 msgError:
     'Si Err=0 no hubo error,
     'si Err<>0 hubo algún error en tiempo de corrida.
     If Err <> 0 Then
     'Mensaje con # de error, descripción del error y # de línea
      MsgBox "Error #: "& Err & Chr(13) & Error & Chr(13) &"Línea " & Erl
     End If
      On Error Goto  0 'reinicializar las variables de error,
                           'es decir Err, Error y Erl.
 End  Function
```

El manejador de errores puede tener cualquier nombre válido, aquí lo llamamos `msgError`, y tiene tres variables: `Err`=número de error, `Error`= descripción del error y `Erl`= número de línea del error en el código.

Raíces $n-$ésimas. A menudo la función a^x se implementa usando la fórmula $a^x = e^{x\ln a}$, $0^x = 1$ (con $0^0 = 1!$); pero no se acepta el caso $a < 0$. Hay que implementar una nueva función `Root(a,n)` para que maneje raíces como $(-8)^{1/3} = \sqrt[3]{-8}$.

Si n es impar y $a < 0$ usamos la fórmula $a^{1/n} = \text{sgn}(a) \cdot |a|^{1/n}$. En otro caso usamos $a^{1/n}$. La función es sencilla de implementar, pero la vamos a usar para mostrar un ejemplo de cómo usar el manejador de errores:

Si n es par, dejamos que se encargue la función default `a^(1/n)`. Si $a < 0$ o $n = 0$, se dispara un error.

Si n es impar, `a^(1/n)=Sgn(a) * Abs(a)^(1/n)` sin importar el signo de a.

Programa 20 `Root(x,n)` *con manejador de errores.*

```
Function  Root(ByVal a As Double, ByVal n As Double) As Double
  Dim m  As Integer
   m = Int(n)    'índice entero
   On Error Goto  msgError  'Manejo de errores
  If m Mod 2 = 0 Then        'Errores posibles: división por cero o
           Root = a ^ (1 / m)    'subradical negativo.
     Else
              Root = Sgn(a) * Abs(a) ^ (1 / m)
  End If
   Exit  Function
```

```
 msgError:
If Err <> 0 Then
MsgBox "Error #: "& Err & Chr(13) & Error & Chr(13) & "En la línea " & Erl
End If
 On Error Goto  0 'reinicializar las variables de error,
                           ' es decir Err, Error y Erl.
End Function
```

A.5.3 Usando la funciones de OOo Calc en OOo Basic.

La lista de funciones disponibles en OOo Calc se puede acceder con `Ctrl F2`. Muchas de estas funciones *no* se encuentran en OOo Basic pero aún así se pueden invocar desde una macro. Usar estas funciones pueden ser muy conveniente porque usualmente están muy bien implementadas.

El nombre a nivel de programación. OOo Calc usa un nombre en la hoja para cada función pero usa otro nombre a nivel de programación. Por ejemplo, en mi versión en español OOo Calc entiende la fórmula `=REDONDEAR(4,4544;2)` pero no entiende la fórmula `=ROUND(4,4544;2)`. Sin embargo, *a nivel de programación* debemos usar el nombre de programación ("programmatic name") e invocar la función `ROUND`.

Para acceder a *todas* las funciones de OOo Calc necesitamos crear un *servicio*. En este caso, el servicio `FunctionAccess`. Este servicio nos provee de propiedades y métodos. Aquí nos interesa el método `callFunction`. El siguiente código general hace una llamada a una función `"NombreFuncion"`. El nombre de la función *siempre es una tira de texto* y el *segundo argumento siempre es un array*

```
Dim oFunction,x
oFunction   = createUnoService("com.sun.star.sheet.FunctionAccess")
         x = oFunction.callFunction("NombreFuncion", Args())
```

En el siguiente ejemplo se muestra la llamada a varias funciones.

Programa 21 *Usando funciones de OOo Calc con el método* `callFunction` *del servicio* `FunctionAccess`.

```
Sub  Main
Dim oFunction, oCell 'Variant
Dim args(1 To 3)   'Array
'Inicializamos el arreglo
 args(1)=4 : args(2)=2.1 : args(3)=0.9
'oCell es la celda A3
 oCell = ThisComponent.Sheets(0).getCellRangeByName("A3")
'Crear el servicio
 oFunction   = createUnoService("com.sun.star.sheet.FunctionAccess")
'Usar el método callFunction del servicio para llamar a la función "SUM"
 oCell.Value = oFunction.callFunction("SUM",Args())

'También es válido usar
 oCell.Value  = oFunction.callFunction("SUM",Array(4,2.1,0.9)) 'Imprime 27
```

```

'Aunque la función sea de un solo argumento, se debe usar un array
 oCell.Value  = oFunction.callFunction("Cos",Array(3.1416))

'Aplicación a Rangos
Dim oRange
'oRange es el rango B4:B6
 oRange     = ThisComponent.sheets(0).getcellrangebyname("B4:B6")
'Aplicamos la prueba Z al rango con  parámetros μ=2.5 y σ=1.
 oCell.Value = oFunction.callFunction("ZTEST", Array(oRange, 2.5, 1.0))
End Sub
```

Nuevas funciones. Podemos incorporar las funciones de OOo Calc como nuevas funciones en nuestra biblioteca `BblMatematica`. Por ejemplo, la función que calcula el valor mínimo de un rango podría ser,

Programa 22 *Función* `MIN` *de OOo Calc.*

```
Function CMIN(oRange)
 Dim oFunction
  oFunction = createUnoService("com.sun.star.sheet.FunctionAccess")
  CMIN = oFunction.callFunction("MIN",oRange)
End Function
```

La podríamos utilizar así,

```
 v = Array(1,2,3,4)
 x = CMIN(v) 'Retorna 1
```

Algunas de las funciones disponibles son,

Tabla A.1 Funciones matemáticas disponibles en OOo Calc

COS	SIN	TAN	COT	ACOS	ACOT	ASIN	ATAN
RADIANS	PICOSH	SINH	TANH	COTH	ACOSH	ACOTH	ASINH
ROUND	ROUNDDOWN	ROUNDUP	CEILING	FLOOR	EVEN	ODD	MROUND
MOD	EXP	POWER	LOG	LN	LOG10	ABS	COMBIN
CONVERT_ADD	COUNTBLANK	COUNTIF	DELTA	ERF	ERFC	FACT	FACTDOUBLE
GESTEP	ISEVEN	ISODD	LCM	LCM_ADD	MULTINOMIAL	PRODUCT	RAND
SIGN	SQRT	SQRTPI	SUBTOTAL	SUM	SUMIF	SUMSQ	SERIESSUM
ATAN2	DEGREES	ATANH	TRUNC	INT	QUOTIENT	COMBINA	CONVERT
BESSELI	BESSELJ	BESSELK	GCD	GCD_ADD	RANDBETWEEN		

En la tabla que sigue se muestran algunas funciones de la hoja para trabajar con matrices. Este tema se desarrolla con detalle en la sección A.5.8.

La lista completa de funciones se puede encontrar en,

`http://wiki.services.openoffice.org/wiki/Documentation/How_Tos/Calc:_Functions_listed_by_category`

Tabla A.2 Funciones para matrices (arrays) en OOo Calc

FREQUENCY	GROWTH	LINEST	LOGEST	MDETERM	MINVERSE	MMULT	MUNIT
SUMPRODUCT	SUMX2MY2	SUMX2PY2	SUMXMY2	TRANSPOSE	TREND		

A.5.4 Un evaluador de funciones matemáticas (“Math Parser”).

La idea aquí es digitar una expresión tal como `x^3+2*x-3*log(x)+1` en una celda y leer y evaluar esta función en algún valor de x. Para hacer esto necesitamos una *evaluador* de funciones matemáticas (“Math parser”). Sin embargo hacer un evaluador requiere el uso de estructuras de datos más complejas de las que queremos usar en este libro. Hay un par de soluciones sencillas: Usar el evaluador de fórmulas de una celda o usar un evaluador de funciones de otro lenguaje, por ejemplo el evaluador de `javascript`.

Cuidado: Este evaluador es muy lento, solo se debería usar para cálculos pequeños. Para cálculos más demandantes es declarar la función directamente en el código.

Un evaluador `Eval(f,x)`. Una manera de evaluar una función es con el método `formula` de las celdas. La estrategia es sencilla: La fórmula de la función es una tira de texto (string), leemos esta expresión y sustituimos la variable por el valor a evaluar y evaluamos la fórmula que nos queda con el método `formula`. Esta manera de proceder requiere una celda que almacene el resultado de la evaluación. Nuestra función `Eval` nos va a permitir hacer cálculos como,

```
f = "2,455*Root(x;3)+x^2"   '"Root" requiere ";" pues evaluamos en la hoja
x = Eval(f,x0)              'Devuelve el valor f(x0). "x" es variable default
f = "t^2+1"
x = Eval(f,t0,"t")          'Devuelve el valor f(t0). "t" es la variable
```

En el siguiente código se muestra como implementar una tal función `Eval(f,x)`. La implementación requiere dos funciones adicionales para manejar `Strings`.

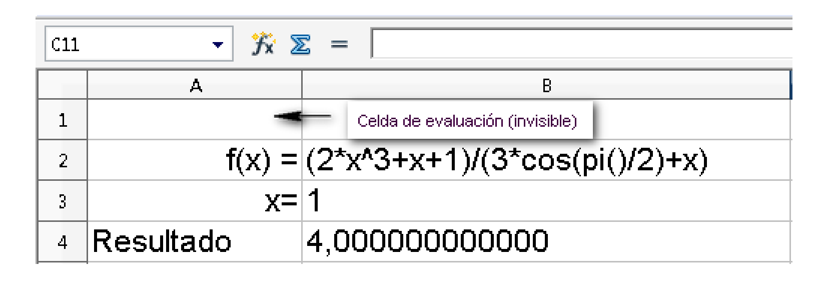


Figura A.20 Usando la función `Eval(f,x)`.

Programa 23 *Evaluador de funciones*

```
'Evalúa funciones usando OOo Calc, por tanto las fórmulas deben ajustarse. Usar solo para cálculos pequeños.
'a la sintaxis de la hoja y a la configuración regional del idioma.
'Evalúa fórmulas funcionales como:"2*x^3-2*cos(x)+log(x+1)", "t^2+3*Exp(t)", etc.
'Puede usar funciones de las bibliotecas: 2,4332*x -3*Root(x;3).
'Observar el uso de "," debido a la configuración regional y el ";".
'Requiere una hoja y una celda auxiliar (i,j) para evaluar (por default es (0,0)).
'La hoja default es la hoja 0
```

```
'Uso: f= oCell.String o f="2*x^2+x+1"
' Eval(f,valor) o Eval(f, valor, "variable")
```

```
Function  Eval(f As String, valor As Double, Optional variable As String,_
                            Optional j, Optional i, Optional numhoja)
    Dim valorVar As String
    Dim nombreVar As String
    Dim formula As String
    Dim CellAuxiliar As Variant
    Dim co, fi,nh, cl
    'Posición de la La celda auxiliar (para evaluar)
    co=0 : fi =0 : nh =0 : nombreVar="x" 'default
    If Not IsMissing(i) And Not IsMissing(j) Then
       co=j : fi =i
    End If
    If Not IsMissing(numhoja) Then
       hn=numhoja
    End If

    If Not IsMissing(variable) Then
           nombreVar = variable
    End If
    CellAuxiliar=ThisComponent.Sheets(nh).getCellByPosition(co,fi)
    'Color de texto = color de fondo
     cl = CellAuxiliar.CellBackColor
     If cl = -1 Then
        CellAuxiliar.CharColor = RGB(255,255,255)
     Else
        CellAuxiliar.CharColor=cl
     End If
    'cambiamos a String para la sustitución
    valorVar= Str(valor)
    formula = f
    'Sustituye  valor  por variable
    Call strSustituir(formula, nombreVar, valorVar)
    ' Las fórmulas inician con "="
    formula="="+formula
    'Evaluar la fórmula
    CellAuxiliar.SetFormula(formula)
 Eval= CellAuxiliar.Value
End Function

'------------------------------------------------------------------
'Subrutina para sustituir
Sub strSustituir( ByRef str1 As String, str2 As String, str3 As String)
Dim i As Long, s1 As String, s2 As String, L1 As Long, L2 As Long
 str1 = " " & str1 & " "
 L2 = Len(str2)
 i = 0
 Do
```

```
    i = InStr(i + 1, str1, str2)
    InStr(i + 1, str1, str2, 1)
   If i = 0 Then  Exit Do
    s1 = Mid(str1, i - 1, 1)
   If Not  IsLetter(s1) Then
        s2 = Mid(str1, i + 1, 1)
        If Not  IsLetter(s2) Then 'Sustituir,
            s1 = Left(str1, i - 1)
            s2 = Right(str1, Len(str1) - i - L2 + 1)
            str1 = s1 & str3 & s2
        End If
    End If
Loop
str1= Trim(str1)
End Sub

Function  IsLetter(ByVal char As String) As Boolean
Dim code As Long
  code = Asc(char)
 IsLetter=(65<=codeAnd code<=90) Or(97<=code And code<=122)Or char="_"
End Function
```

Programa 24 *Usando el evaluador en Main*

```
Sub  Main
 'Usando la función Eval(f,x) para evaluar f en 'valor'
  'La fórmula debe respetar la sintaxis permitida en la HOJA
Dim f, valor
'Leemos la fórmula en la celda "B2"
 f = ThisComponent.Sheets(0).getCellByPosition(1,1).getString()
'Leemos el valor de "x"
 valor =ThisComponent.Sheets(0).getCellByPosition(1,2).Value
 'Evaluamos con Eval(f,valor). "x" es la variable default
 ThisComponent.Sheets(0).getCellByPosition(1,3).Value= Eval(f,valor)

 'También puede introducir la fórmulas directamente,
 f="2,455*Root(t;3)+t^2" '"Root" es de nuestra biblioteca BblMatematica
 MsgBox  Eval(f,valor,"t")
End Sub
```

Evaluar funciones de dos variables. La misma idea del evaluador para funciones de una variable se puede usar en dos variables: Sustituimos cada variable una a la vez y luego evaluamos. La función la llamamos `Eval2`, las variables default son x e y.

Nuestra función `Eval2` nos va a permitir hacer cálculos como,

```
 f = "x^2+y^2+1"
 MsgBox  Eval(f,x0,y0) '"x" e "y" son variables default, en ese orden.
```

```
f = "t^2+y^2+1"
MsgBox  Eval(f,t0,y0,"t","y")
```

Programa 25 *Eval2 para evaluar funciones de dos variables*

```
' Eval2 = evalúa una función de dos variables.
' Eval(fxy,x0,y0) -> f(x0,y0)
' Eval2(ftw,t0,w0,"t","w") -> ftw(t0,w0)
```

```
Function Eval2(f As String, valor1 As Double, valor2 As Double,_
            Optional var1 As String,  Optional var2 As String,_
             Optional j, Optional i, Optional numhoja)
    Dim valorVar1 As String
    Dim valorVar2 As String
    Dim formula As String
    Dim nombreVar1 As String
    Dim nombreVar2 As String
    Dim CellAuxiliar As Variant
    Dim co, fi,nh, cl
    'Posición de la La celda auxiliar (para evaluar)
    co=0 : fi =0 : nh =0 : nombreVar1="x" : nombreVar2="y" 'default
    If Not IsMissing(i) And Not IsMissing(j) Then
       co=j : fi =i
   End If
    If Not IsMissing(numhoja) Then
       hn=numhoja
    End If

    If Not IsMissing(var1) Then
       nombreVar1=var1
    End If
    If Not IsMissing(var2) Then
       nombreVar2=var2
    End If
    CellAuxiliar=ThisComponent.Sheets(nh).getCellByPosition(co,fi)
    'Color de texto = color de fondo
     cl = CellAuxiliar.CellBackColor
     If cl = -1 Then
        CellAuxiliar.CharColor = RGB(255,255,255)
     Else
        CellAuxiliar.CharColor=cl
     End If
    'cambiamos a String para la sustitución
    valorVar1 = Str(valor1) :  valorVar2 = Str(valor2)
    formula = f
    'Sustituye  valorVari por nombreVari
    Call strSustituir(formula, nombreVar1, valorVar1)
    Call strSustituir(formula, nombreVar2, valorVar2)
    ' Las fórmulas inician con "="
    formula="="+formula
    'Evaluar la fórmula
```

```
    CellAuxiliar.SetFormula(formula)
 Eval2= CellAuxiliar.Value
End Function
```

A.5.5 Vectores, matrices y rangos.

Los "arreglos" (`array`) son de uso frecuente no solo en cálculos matriciales, muchos cálculos requieren este formato, por ejemplo en interpolación donde se debe seleccionar un rango de datos.

Arrays. Un `array` ("arreglo") es una estructura de datos que se usa para indexar datos. Por ejemplo columnas, filas o tablas de números. Es obligatorio declarar los arreglos antes de usarlos. Para declarar un arreglo se usa `Dim` y se usa paréntesis para definir y accesar los elementos del arreglo. En la tabla (A.5.5) se muestra la manera de definir un arreglo y una breve descripción.

Declaración	Elementos	Descripción
`Dim v(n) As Integer`	$n+1$	Enteros $v(0),v(1),...,v(n)$
`Dim v(1 To n) As Integer`	n	Enteros $v(1),v(2),...,v(n)$
`Dim w(-n To n) As Double`	$2n+1$	$w(-n),w(-n+1),...,w(0),w(1),...,w(n)$
`Dim M(n, m)`	$(n+1)\times(m+1)$	$M(i,j)=a_{ij},\ \begin{pmatrix} a_{00} & a_{01} & \cdots & a_{0,m} \\ a_{10} & a_{11} & \cdots & a_{1,m} \\ & & \vdots & \\ a_{n0} & a_{n1} & \cdots & a_{n,m} \end{pmatrix}$
`Dim M(1 To n,1 To m)`	$n\times m$	$M(i,j)=a_{ij},\ \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1,m} \\ a_{21} & a_{22} & \cdots & a_{2,m} \\ & & \vdots & \\ a_{n1} & a_{n1} & \cdots & a_{n,m} \end{pmatrix}$

Tabla A.3

Un arreglo se puede declarar de manera directa, por ejemplo

```
Dim v()
v=Array(cos(1),2.3,4.2)    'v(0)=cos(1), v(1)=2.3 y v(2)=4.2
```

Aunque podemos crear una tabla como un arreglo "de arreglos", es mejor hacerlo de manera directa, por ejemplo,

```
Dim v(0 To 1, 0 To 2)
 v(0, 0) = 1 : v(0, 1) = 2 : v(0, 2) = 3
 v(1, 0) = 3 : v(1, 1) = 5 : v(1, 2) = 2.3
```

ReDim. Es frecuente no conocer las dimensiones del arreglo antes de seleccionar un rango o leer una o más variables. Podemos declarar el arreglo sin dimensiones y redimensionarlo en cuando tengamos el dato, por ejemplo,

```
Dim v()
...
ReDim v(1 To n)
```

LBound, UBound. Para recorrer un vector `w` de tamaño desconocido podemos usar `LBound(w)` y `UBound(w)` para tener acceso al primer y último subíndice, respectivamente, del arreglo.

También podemos leer las dimensiones de una arreglo 2-dimensional (una matriz) A. Si se declara `ReDim A(f1 to fn, c1 to cn)`, entonces `LBound(A,1)` devuelve `f1`, `UBound(mA,1)` devuelve `fn`, `LBound(A,2)` devuelve `c1` y `UBound(mA,2)` devuelve `cn`. Por ejemplo,

Programa 26 *Usando arreglos*

```
Sub  Main
Dim i
Dim w()
ReDim w(1 To n)
...
'La primera entrada de w es w(LBound(w))  y la última es w(UBound(w))
For i=LBound(w) To  UBound(w)
    w(i)=0
Next i

Dim A()
ReDim A(10,15) '11 filas, 16 columnas

MsgBox LBound(mA,1)  '-> 0  (fila 0)
MsgBox UBound(mA,1)  '-> 10 (fila 10)
MsgBox LBound(mA,2)  '-> 0  (columna 0)
MsgBox UBound(mA,2)  '-> 15 (columna 15)
End Sub
```

Preserve. Podemos aumentar la dimensión del arreglo conservado los datos ya alamcenados en el arreglo. Esto se hace con `ReDim Preserve A`. Por ejemplo,

Programa 27 *Usando* `Preserve`

```
Sub  Main
Dim i
Dim A()
ReDim A(1 To 2, 1 To 2)

   A(1,1)=11 : A(1,2)=12 : A(2,1)=21 : A(2,2)=22 'A = 11  12
                                                '    21  22
ReDim Preserve A(2,2)
                                                '    _   _   _
                                                'A = _   11  12
                                                '    _   21  22
End Sub
```

Array de Array's. A veces tenemos una matriz como un un array con array's. Lo bueno de esto es que las filas se pueden accesar como vectores, por ejemplo

Programa 28 *Arreglo de arreglos*

```
Sub  Main
Dim T()
Dim fila
  T = Array(Array(3,5,7),Array(4,5,6))
  fila = T(0)     '3 5 7
MsgBox fila(2) '-> 7
  fila = T(1)     '4 5 6
MsgBox fila(0) '-> 4
End Sub
```

También podemos escribir,

Programa 29 *Arreglo de arreglos. Otra manera.*

```
Sub  Main
Dim T()
Dim fila
  T(0)=Array(3,5,7)
  T(1)=Array(4,5,6))
  fila = T(0)     '3 5 7
MsgBox fila(2) '-> 7
  fila = T(1)     '4 5 6
MsgBox fila(0) '-> 4
End Sub
```

A.5.6 Funciones que reciben o devuelven arreglos.

Una función puede recibir arreglos como argumentos y entregar números o arreglos. El arreglo que recibe se declara Variant.

En el código que sigue se implementa la norma de un vector v y el producto escalar. Si $v = (v_1, v_2, ..., v_n)$ su norma es $||v|| = \sqrt{v_1^2 + v_2^2 + ... + v_n^2}$ y $k \cdot v = (k \cdot v_1, k \cdot v_2, ..., k \cdot v_n)$.

Programa 30 *Norma y producto escalar*

```
Function Norma(V() As Variant)
  Dim suma,i
```

```
   suma = 0
  For  i=LBound(V) To UBound(V)
      suma =suma+V(i)^2
  Next i
   Norma = Sqr(suma)
End Function

Function Escalar(V(), k )
  Dim W()
  Dim i,n

  n = UBound(V)
 ReDim W(n)
 For  i=LBound(V) To UBound(V)
   'V no se modifica
    W(i) = k*V(i)
 Next i
 Escalar = W()
End Function
```

Un ejemplo de su uso es,

Programa 31 *Cálculos con la norma y el producto escalar*

```
Sub  Main
  Dim x
  Dim B()
  Dim S()

  B = Array(0,3,4)
  x = Norma(B)         'Retorna 5
  S = Escalar(B, 2)    'Retorna (0,6,8)
End Sub
```

A.5.7 Rangos.

Una operación frecuente es seleccionar con el ratón un rango, contar las filas y aplicar alguna fórmula. La selección del usuario se puede almacenar en una variable `rango` y contar las filas de la selección, de la siguiente manera,

```
Dim rango
Dim n
 rango = ThisComponent.getCurrentSelection()
 n     = rango.Rows.getCount()
```

Como antes, se puede accesar el rango por columnas y filas,

```
rango.getCellByPosition(0, i).Value
```

nos devuelve el valor en la primera columna del rango (columna 0) y en la fila i (contando desde cero). El valor en columna `j` del rango y en la fila `i` (ambos, contando desde cero), se obtiene con

```
rango = ThisComponent.getCurrentSelection()
rango.getCellByPosition(j, i).Value
```

Estadística básica. Supongamos que tenemos una muestra de N datos $x_0, x_1, ..., x_{N-1}$ de una población. El promedio muestral $\overline{x} = (x_0 + x_1 + \cdots + x_{N-1})/N$ es una estimación del promedio poblacional μ (el valor esperado del promedio de todas las muestras). La varianza poblacional σ^2 se estima con la varianza muestral $s^2 = \sum_{i=0}^{N-1}(x_i - \overline{x})/N$. La desviación estándar de la población se estima con s. Otros valores de interés son el valor máximo y el valor mínimo y la mediana.

En el cuaderno que se muestra en la figura (A.21) hay una columna con los datos x_i. El botón `Estadísticas` ejecuta la subrutina `Estadisticas()`. En esta subrutina se lee el rango seleccionado por el usuario y se calcula el máximo, la media muestral, la varianza muestral y la desviación estándar muestral.

B	C	D	E	F
	Estadísticas			
x_i	Máximo	Media	Varianza	Desviacion Std
76,56	93,75	79,68	154,88	12,44515
78,12				
87,5				
85,93				
87,5				
73,43				
85,93				
93,75				
87,5				
62,5				
60,93				
51,56				
68,75				
84,37				
85,62				
95,31				
62,5				

Figura A.21 Estadística básica

Programa 32 *Estadística básica*

```
Sub Estadisticas()
  Dim n, i
  Dim Rango
  Dim X()
  Rango = ThisComponent.getCurrentSelection()
      n = Rango.Rows.getCount()
  ReDim X(0 To n-1)
  For i=0 To n-1
    X(i)=Rango.getCellByPosition(0,i).Value
```

```
  Next i
  ThisComponent.Sheets(0).getCellRangeByName("C3").Value = MaxVect(X)
  ThisComponent.Sheets(0).getCellRangeByName("D3").Value = Media(X)
  ThisComponent.Sheets(0).getCellRangeByName("E3").Value = Var(X)
  ThisComponent.Sheets(0).getCellRangeByName("F3").Value = DevStd(X)
End Sub
'-----------------------------------------------------------------
Function MaxVect(X())
 Dim mx, im
  mx = X(0)   : im = UBound(X)

   For i = 0 To im
    If X(i) > mx Then
        mx = X(i)
    End If
  Next i
 MaxVect = mx
End Function

Function Media(X())
 Dim suma, i0,im
   suma = 0 : im = UBound(X) '# datos = im+1

  For i = 0 To im
         suma = suma + X(i)
  Next i
 Media  = suma/(im+1)
End Function

Function Var(X())
 Dim promedio, suma, im
  promedio = Media(X): suma = 0  : im = UBound(X) '# datos = im+1

  For i = 0 To im
         suma = suma +(X(i)-promedio)^2
  Next i
 Var  = suma/im
End Function

Function DevStd(X())
         DevStd = Sqr(Var(X))
End Function

```

En el programa anterior se pasó los valores del rango a un vector. Aunque no es necesario, a veces es sumamente cómodo hacer esto porque las fórmulas y los algoritmos son más fáciles de seguir y depurar.

Por ejemplo, En el siguiente código pasamos los valores de la primera columna (columna 0) a un vector `X()` y pasamos los valores de la segunda columna (columna 1, suponiendo que el rango tiene dos o más columnas) a un vector `Y()`,

Programa 33 *Pasar las columnas de un rango a vectores*

```
Dim rango
Dim n, i
Dim X(), Y()
 rango = ThisComponent.getCurrentSelection()
 n     = rango.Rows.getCount()

ReDim X(1 To n)
ReDim Y(1 To n)

For i=1 To n
   X(i)=rango.getCellByPosition(0, i-1).Value
   Y(i)=rango.getCellByPosition(1, i-1).Value
Next i
```

En el código que sigue, se pasa la totalidad de una rango a una matriz `A`.

Programa 34 *Pasar una rango a una matriz*

```
Dim rango
Dim n,m, i,j
Dim Y()
Dim A()
 rango = ThisComponent.getCurrentSelection()
 n     = rango.Rows.getCount()
 m     = rango.Columns.getCount()
ReDim A(1 To n, 1 To m) 'Filas x columnas!, A=(a_ij), i=1..n,j=1..n
For i=1 To n
   For j=1 To m
           A(i,j)=rango.getCellByPosition(j-1, i-1).Value
   Next j
Next i
```

Con solo que haya una celda seleccionada, ya el rango tendrá al menos una fila y una columna. En la práctica, muchas de las operaciones de rango requieren que la selección tenga dos o más filas (o columnas). Si este es el caso, se puede agregar una instrucción que envíe un mensaje si no se ha seleccionado la cantidad mínima de datos y además salir de la función o la subrutina (pues no habría nada que hacer!). Para este propósito podemos usar el código

Programa 35 *Mensaje para advertir sobre la cantidad de datos seleccionados.*

```
Sub  SeleccionarDatos()
   Dim rango
   Dim n
```

```
    rango = ThisComponent.getCurrentSelection()
    n     = rango.Rows.getCount()

If n<=1 Then
     MsgBox  "Por favor, seleccione los datos."
     Exit Sub
End If
    ...
End Sub
```

Hacer una copia de una Matriz. Si tenemos dos matrices `M1()` y `M2()`, para hacer una copia en `B` de `A`, lo mejor es usar un ciclo `For` y hacer la copia componente a componente:

Programa 36 *Subrutina para hacer una copia B de una matriz A*

```
Sub MCopiar(B(),A())
  Dim f1, fn, c1, cm, i, j
  f1   = LBound(A,1)   ' primera fila
  fn =   UBound(A,1)   ' última fila
  c1 =   LBound(A,2)   ' primera columna
  cm =   UBound(A,2)   ' última columna
  ReDim B(f1 To fn, c1 To cm )

  For i=f1 To fn
    For  j=c1 To cm
        B(i,j)=A(i,j)
   Next j
 Next i
End Sub
```

La razón de hacer esto así es porque la asignación `B()= A()` hace que estas dos matrices queden vinculadas, es decir, los cambios en `A` se reflejan en `B` y viceversa. Para algunos cálculos esto no es nada conveniente.

Programa 37 *Probando la subrutina* `MCopiar`

```
Sub  Main
  Dim A()
  Dim B()
  Dim C()
  Dim i,j
  ReDim A(1 To 5, 1 To 5)
  For i=1  To 5
     For  j=1  To 5
        A(i,j) = 0
     Next j
  Next i
 MCopiar(B,A)   'B es una copia de A
 C = A          'C está vinculada con A
```

```
 A(1,1)= 1
 MsgBox A(1,1) '-> 1
 MsgBox C(1,1) '-> 1, C cambió igual que A
 MsgBox B(1,1) '-> 0, B no cambia con A
End Sub
```

A.5.8 Funciones para operaciones con matrices.

Como ya vimos en la sección (A.5.3), podemos implementar funciones que usen las funciones para el manejo de rangos de la hoja OOo Calc. Puede ser bueno usar estas funciones porque generalmente son muy bien implementadas. Otras funciones las tendremos que implementar según los requerimientos de los algoritmos que estudiemos en el futuro.

Función Det(A). En el código que sigue, implementamos una función `CDET(A)` para calcular el determinante de la matriz $A_{n\times n}$. Usamos la función $MDETERM$ de la hoja. Solo hay que recordar que la matriz A se debe recibir como un array, es decir, como `Array(A)`.

Programa 38 *Función determinante usando la función* `MDETERM` *de OOo Calc*

```
Function CMATRIXDET(oRange)
 Dim oFunction
   oFunction = createUnoService("com.sun.star.sheet.FunctionAccess")
   CMATRIXDET = oFunction.callFunction("MDETERM",Array(oRange))
End Function
```

Un ejemplo de su uso se muestra en el código,

Programa 39 *Usando la función* `CMATRIXDET`

```
Sub  Main
  Dim A() As Double
  Dim x
  ReDim A(1 To 2,1 To 2)
    A(1,1)= 2 : A(1,2) = 2 : A(2,1)= 3 : A(2,2) = -3
    x = CMATRIXDET(A)        'Retorna -12

  'oRange es un rango en la hoja 0
  Dim oRange
      oRange = ThisComponent.sheets(0).getcellrangebyname("C6:D7")
            x = CMATRIXDET(oRange)
End Sub
```

Funciones que devuelven arreglos anidados. Hay que hacer algunos arreglos si la función de OOo Calc devuelve una array anidado (Ver [34]), como es el caso de la multiplicación de matrices y el cálculo de la inversa. En este caso lo que devuelve la función de OOo Calc es un array del tipo

`Array(filas(columnas(0 to n-1))),`

por lo que debemos *recuperar los datos por filas*. Por ejemplo,

Programa 40 *Inversa de una matriz, primera versión.*

```
Function  CMATRIXINVERSE(oRange)
 Dim Fila, Filas, mM, nf, i, j
 Dim oFunction
  oFunction = createUnoService("com.sun.star.sheet.FunctionAccess")
  Filas = oFunction.callFunction("MINVERSE",Array(oRange))
  '"Filas" es un array anidado
  'nf = número de filas
 nf = UBound(Filas)
 ReDim mM(0 To nf, 0 To nf)
 For i = 0 To nf
     Fila = Filas(i)
     For j = 0 To nf
        mM(j,i)= Fila(j)
     Next j
  Next i
 CMATRIXINVERSE= mM
End Function
```

En vez de poner todo este código, podemos proceder como en (Ver [34]) para editar de una manera más limpia las funciones que invocan funciones de OOoCalc que devuelven un array anidado (y también para las otras). Se necesitan dos funciones, la primera es `calc_Func` para llamar la función de OOo Calc y la otra es `DataArray2PlainArray` que lo que hace es convertir el array anidado en un array corriente de dimensiones `(1 to n, 1 to m)`. En el ejemplo que sigue, primero presentamos una subrutina `Test()` que usa las funciones implementadas más abajo (las cuales agregamos a nuestra biblioteca `BblMatematica`!).

Programa 41 *Operaciones con matrices usando las funciones de OOo Calc*

```
Sub  Test()
  Dim i,j
  Dim mA(2)
  mA(0) = Array( 1, 2 )
  mA(1) = Array( 4, 5 )
  mA(2) = Array( 7, 8 )
 Dim mB(1)
  mB(0) = Array ( 3, 2, 1 )
  mB(1) = Array ( 6, 5, 4 )
 Dim mC()
 'Cálculo de la transpuesta de mA
  mC = CTRANSPOSE(mA)
 For i=1 To UBound(mC,2)
   For j = 1 To UBound(mC,1)
      MsgBox mC(j,i)
```

```
    Next j
   Next i
   'Cálculo de mA·mB
   mC=CMATRIXMULTIPLICATION(mA,mB)
   For i=1 To UBound(mC,2)
     For j = 1 To UBound(mC,1)
       MsgBox mC(j,i)
    Next j
  Next i
End Sub
'-----------------------------------------------------------------
'Función para llamar la función 'MMULT' (multiplicación matricial) de la hoja
Function  CMATRIXMULTIPLICATION(A(), B())
 Dim  mM()
    mM = calc_Func("MMULT",Array(A(),B()))
    DataArray2PlainArray mM()
 CMATRIXMULTIPLICATION = mM
End Function

'Función para llamar la función 'TRANSPOSE' de la hoja
Function  CTRANSPOSE(A())
  Dim  mM()
    mM = calc_Func("TRANSPOSE",Array(A()))
    DataArray2PlainArray mM()
 CTRANSPOSE = mM
End Function

'Función para llamar las funciones de la hoja vía el servicio 'FunctionAccess'
Function  calc_Func(sFunc$,args())
 Dim oFA As Object
   oFA = createUNOService("com.sun.star.sheet.FunctionAccess")
 calc_Func = oFA.callFunction(sFunc,args())
End Function

'Cambiar array anidado a array corriente
Sub  DataArray2PlainArray(aRows())
 Dim i%,j%, aCols(),aTmp()
 ReDim aTmp(1 To UBound(aRows())+ 1,  1 To UBound(aRows(0)) +1)
 For i = 0 To UBound(aRows())
    aCols = aRows(i)
    For j = 0 To UBound(aCols())
       aTmp(i +1,j +1) = aCols(j)
    Next
Next
  aRows() = aTmp()
End Sub
```

A.6 Bibliotecas especiales.

Hasta a hora hemos implementado algunas funciones y subrutinas de ejemplo. En ingeniería y ciencias se necesitan bibliotecas de funciones espaciales: De análisis de datos, métodos numéricos, etc. La primera biblioteca que vamos a implementar es una biblioteca con funciones y subrutinas generales de uso frecuente. Por supuesto, las primeras funciones que podemos agregar son las funciones que ya hemos implementado más arriba. Además de estas funciones, necesitamos algunas funciones adicionales. Recordemos que este capítulo corresponde al capítulo introductorio de un curso de métodos numéricos, así que que las nuevas funciones de la bibliotecas irán apareciendo en el camino.

A.6.1 Biblioteca BblMatematica de funciones de uso frecuente.

Esta biblioteca contiene funciones especiales y funciones misceláneas. Lo que hacemos es abrir un cuaderno nuevo, crear la biblioteca y luego exportarla. Luego podemos usar este mismo cuaderno o importar la biblioteca. En todo caso, antes de usar las funciones de la biblioteca, hay que cargarla en la subrutina Main,

BblMatematica_Version 1..ods.Standard - OpenOffice.org Basic

Archivo Editar Ver Herramientas Ventana Ayuda

[BblMatematica_Version 1..ods].Standard

```
Option Explicit
Sub Main
 'Cargar la biblioteca BblMatematica
  BasicLibraries.loadLibrary("BblMatematica" )
End Sub
'----------------------------------------------
Function DDNewton(X(),Y())
Dim i, dx
```

Figura A.22 Importar y cargar la biblioteca BblMatematica.

Ahora, podemos agregar varias de las funciones y subrutinas que hemos implementado a esta biblioteca.

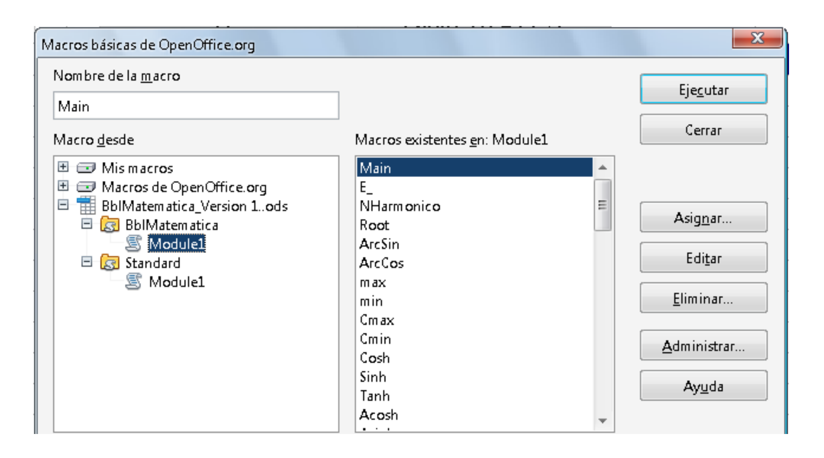


Figura A.23 Biblioteca BblMatematica.

A.6.2 Algunas funciones especiales

Raíces $n-$ésimas. Si n es impar y $a < 0$ usamos la fórmula $a^{1/n} = \text{sgn}(a) \cdot |a|^{1/n}$. En otro caso usamos $a^{1/n}$.

Programa 42 `Root(x,n)` *con manejador de errores.*

```
Function  Root(ByVal a As Double, ByVal n As Double) As Double
  Dim m  As Integer
   m = Int(n)    'índice entero
   On Error Goto  msgError  'Manejo de errores
  If m Mod 2 = 0 Then        'Errores posibles: división por cero o
           Root = a ^ (1 / m)    'subradical negativo.
     Else
              Root = Sgn(a) * Abs(a) ^ (1 / m)
  End If
   Exit  Function
   msgError:
  If Err <> 0 Then
  MsgBox "Error #: "& Err & Chr(13) & Error & Chr(13) & "En la línea " & Erl
  End If
   On Error Goto  0 'reinicializar las variables de error,
                                 ' es decir Err, Error y Erl.
End Function
```

Funciones trigonométricas inversas. En OOo Basic tenemos únicamente las funciones $\cos(x)$, $\text{sen}(x)$, $\tan(x)$ y $\arctan(x)$. El resto de funciones trigonométricas se deben implementar usando éstas o usando las funciones de la hoja. Muchas fórmulas se pueden encontrar en libros de tablas y fórmulas matemáticas como [26].

ArcSen(x) y ArcCos(x). Para implementar estas funciones necesitamos una identidad que las relacione con $arctan(x)$. Consideremos la figura (A.24),

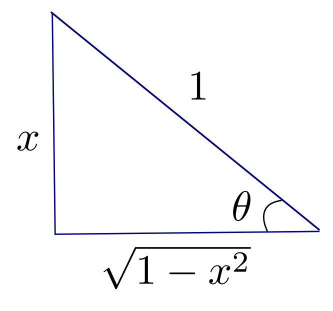


Figura A.24

Como $\text{sen}\,\theta = x$ entonces $\text{arcsen}(x) = \arctan\left(\frac{x}{\sqrt{1-x^2}}\right)$, si $0 < x < 1$. Como $-\text{arcsen}\,x = \text{arcsen}(-x)$, $-\arctan x = \arctan(-x)$ y $\arccos x = \pi/2 - \text{arcsen}\,x$, entonces

$$\begin{aligned} \text{arcsen}(x) &= \arctan\left(\frac{x}{\sqrt{1-x^2}}\right) \quad \text{si} -1 < x < 1, \\ \text{arccos}(x) &= \arctan\left(\frac{-x}{\sqrt{1-x^2}}\right) + \pi/2 \quad \text{si} -1 < x < 1. \end{aligned}$$

Programa 43 *Funciones* `arcsen(x)` *y* `arccos(x)`

```
Function Acos(ByVal a As Double) As Double
  If a = 1 Then
    Acos = 0
  Else If a = -1 Then
    Acos = PI
  Else
    Acos = Atn(-a / Sqr(-a * a + 1)) + PI / 2
  End If
End Function

Function Asin(ByVal a As Double) As Double
  If Abs(a) = 1 Then
    Asin= Sgn(a) * PI / 2
  Else
    Asin = Atn(a / Sqr(a * a + 1))
  End If
End Function
```

Min, Max. Mínimo y máximo de dos valores.

Programa 44 *Funciones* `Min` *y* `Max`

```
Function Min( p1, p2 )
   If p1 < p2 Then
      Min = p1
   Else
      Min = p2
   End If
End Function

Function Max( p1, p2 )
   If p1 > p2 Then
        Max = p1
   Else
     Max = p2
   End If
End Function
```

Random entero entre `nMin` y `nMax`. `Rnd()` devuelve un número 'aleatorio' entre 0 y 1. Para generar enteros 'aleatorios' entre `nMin` y `nMax` se usa la fórmula

`Int(nMin + RND() * (nMax - nMin))).`

Por ejemplo, para generar enteros aleatorios entre 10 y 30 se usa `Int(10+Rnd() * 20).`

Programa 45 *Número entero aleatorio entre* `nMin` *y* `nMax`*.*

```
Function IntRandom( ByVal nMin As Double, ByVal nMax As Double ) As Double
   IntRandom= Int(nMin + RND() * (nMax - nMin))
End Function
```

Función Gamma y factorial. La función Gamma se define como

$$\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}dt$$

Una de las propiedades importantes de esta función es

$$\Gamma(z+1) = z\Gamma(z)$$

En particular, como $\Gamma(1) = 1$ entonces $\Gamma(n+1) = n!$ si $n \in Z$. Esta última fórmula no es adecuada para calcular $n!$ puesto que este caso particular de la función Γ se puede calcular de manera más sencilla.

La función Γ se calcula usualmente con la aproximación (Lanczos, 1964. [**?**, pág 256]),

$$\Gamma(z+1) = (z+\gamma+\frac{1}{2})^{z+\frac{1}{2}}e^{-(z+\gamma+\frac{1}{2})} \cdot \sqrt{2\pi}\left(c_0 + \frac{c_1}{z+1} + \frac{c_2}{z+2} + \cdots + \frac{c_N}{z+N} + \epsilon\right) \quad \text{si} \quad z > 0.$$

La fórmula también es válida si $z \in \mathbb{C}$ y $\text{Re}\, z > 0$. P. Godfrey calculó un conjunto de 14 coeficientes a_i con los cuales el error $|\epsilon| < 10^{-15}$, suficiente para nuestros cálculos en el computador.

Es conveniente calcular $\ln\Gamma(z)$ para evitar los desbordamientos tempranos (overflow). Sea $A = z+\gamma+\frac{1}{2}$, entonces

$$\begin{aligned} \ln\Gamma(z) &= \ln\left(\frac{\Gamma(z+1)}{z}\right) \\ &\approx \frac{(z+0.5)\ln A - A + \ln\left[\sqrt{2\pi}\left(c_0 + \frac{c_1}{z+1} + \frac{c_2}{z+2} + \cdots + \frac{c_14}{z+14}\right)\right]}{z} \end{aligned}$$

donde, $a_0 = 0.999999999999997092$, $\gamma = 671/128$ y

$$
\begin{aligned}
a_1 &= 57.1562356658629235 \\
a_2 &= -59.5979603554754912 \\
a_3 &= 14.1360979747417471 \\
a_4 &= -0.491913816097620199 \\
a_5 &= 0.339946499848118887 \times 10^{-4} \\
a_6 &= 0.465236289270485756 \times 10^{-4} \\
a_7 &= -0.983744753048795646 \times 10^{-4} \\
a_8 &= 0.158088703224912494 \times 10^{-3} \\
a_9 &= -0.210264441724104883 \times 10^{-3} \\
a_{10} &= 0.217439618115212643 \times 10^{-3} \\
a_{11} &= -0.164318106536763890 \times 10^{-3} \\
a_{12} &= 0.844182239838527433 \times 10^{-4} \\
a_{13} &= -0.261908384015814087 \times 10^{-4} \\
a_{14} &= 0.368991826595316234 \times 10^{-5}
\end{aligned}
$$

Programa 46 *Función* $\ln(\Gamma(x))$ *(aproximación de Lanczos).*

```
Function  LnGamma(x As Double)
 Dim tmp As Double
 Dim suma As Double
 Dim Cf(14)
 Dim i
    Cf(0) = 0.999999999999997
    Cf(1)=57.1562356658629235
    Cf(2)=-59.5979603554754912
    Cf(3)=14.1360979747417471
    Cf(4)=-0.491913816097620199
    Cf(5)=0.339946499848118887E-4
    Cf(6)=0.465236289270485756E-4
    Cf(7)=-0.983744753048795646E-4
    Cf(8)=0.158088703224912494E-3
    Cf(9)=-0.210264441724104883E-3
    Cf(10)=0.217439618115212643E-3
    Cf(11)=-0.164318106536763890E-3
    Cf(12)=0.844182239838527433E-4
    Cf(13)=-0.261908384015814087E-4
    Cf(14)=0.368991826595316234E-5
    tmp = x + 5.2421875      'x+γ+1/2, γ =671/128
    tmp =(x+0.5)*Log(tmp)-tmp
    suma = Cf(0)
    For i = 1 To 14
        suma = suma + Cf(i) / (x + i)
    Next i
     LnGamma = tmp+Log(2.5066282746310005*suma/x)
End Function
```

Para calcular usamos `Exp(LnGamma(x))` con $x < 172$.

```
For i=1  To 5                '2,67893853470775
    MsgBox Exp(LnGamma(i/3))  '1,3541179394264
Next i                       '1
                             '0,89297951156925
                             '0,902745292950934
MsgBox Exp(LnGamma(171.6))    '1,58589690966708E+308
MsgBox Exp(LnGamma(172))      '-> "Desbordamiento"
```

La función factorial la habíamos implementado antes,

Programa 47 *La función factorial.*

```
Function Factorial(n)
Dim i
Dim producto As Double
    producto=1
   For  i= 2 To n
        producto = producto*i
    Next i
    Factorial = producto
End Function
```

Se puede calcular `factorial(n)` con $1 \leq n < 171$ pero solo es exacta hasta $n = 20$, pues $20! = 243290200817664$ (15 dígitos) y `factorial(20)=2,43290200817664E+018`. Para valores más grandes de n el computador recurre a una aproximación, por ejemplo $21! = 5109094217170944$ (16 dígitos) mientras que `factorial(21)=5,10909421717094E+019`.

A.6.3 Funciones y subrutinas misceláneas

Una función `Cells`. La acción de leer y escribir en una celda es muy frecuente. Es conveniente tener un par de funciones para simplificar la lectura y la escritura.

Programa 48 *Funciones para celdas*

```
Function Cells(txt  As String, Optional numhoja)
  Dim nh
  If Not IsMissing(numhoja) Then
      nh = numhoja
  Else nh=0
  End If
  Cells = ThisComponent.Sheets(nh).getCellRangeByName(txt)
```

```
End Function

Function CellsCF(columna, fila,  Optional numhoja)
  Dim nh
  If Not IsMissing(numhoja) Then
      nh = numhoja
  Else nh=0
  End If
   CellsCF = thisComponent.Sheets(nh).getCellByPosition(columna, fila)
End Function
```

La función `Cells` se puede usar para leer y escribir en las celdas de una hoja usando el nombre de la celda. La hoja default es la hoja 1. Por ejemplo, para leer la celda `A5` de la hoja 1 escribimos

```
Dim x
x = Cells("A4").Value.
```

La función `CellsCF` se puede usar para lo mismo haciendo referencia a la posición de la celda. Por ejemplo, para escribir el valor `f(x)` en la celda `B5` de la hoja 1, escribimos

```
CellsCF(1,4).Value=f(x)
```

Una función para limpiar un rango. Hay cálculos que llenan un rango. Un cálculo posterior puede llenar un rango menor y causar confusiones entre los datos nuevos y los viejos. Una manera de evitar este problema es implementando una subrutina que *limpie* las celdas (ver figura A.25).

La subrutina `CleanRange(co,fi,nc)` limpia un rango iniciando en la celda (co, fi). Desde esta celda baja borrando `nc` columnas a la derecha, hasta que se encuentre una celda en blanco, es decir, una celda con cero caracteres.

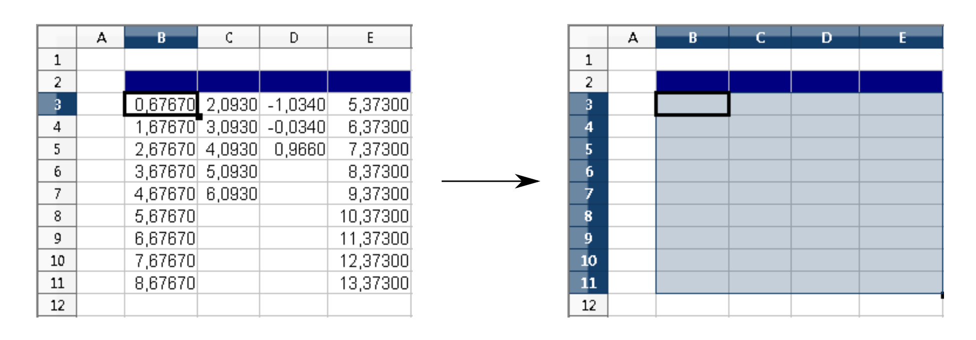


Figura A.25 `Call CleanRange(1,2,4).`

Programa 49 *Subrutina para limpiar un rango*

```
'Limpia Rango
'cini       = columna en inicia la limpieza
'fini       = fila en que inicia la limpieza
'nc         = número de columnas (a la derecha) a borrar desde cini inclusive
```

```
'numhoja es el número de hoja, valor default = 0 (hoja 1).
Sub CleanRange(cini,fini, nc, Optional numhoja)
 Dim k,j, nh, lg
 Dim oCell, Hoja

 If Not IsMissing(numhoja) Then
       nh = numhoja
 Else nh=0
 End If
  Hoja = thisComponent.Sheets(nh)
  k=0
 Do
              'Recorremos la columna cini
       OCell = Hoja.getCellByPosition(cini, fini+k)
       lg = Len(oCell.String)
              'lg = 0 si la celda está en blanco
     If lg <> 0 Then
      For j = 0 To nc
             'Borra la fila "nc" columnas a la derecha
                Hoja.getCellByPosition(cini+j, fini+k).setString("")
      Next j
    End If
       k = k + 1
            'Hasta que encuentre la primera celda en blanco
Loop Until lg=0
End Sub
```

A.7 Gráficos.

El propósito de esta sección es desplegar la representación gráfica de una función desde una subrutina. Con OOo Basic se puede mostrar datos en forma de "diagramas" (gráficos de barras, pie, etc.), es decir, se crean vínculos gráficos con los datos, en forma de barras, sectores, líneas, etc.

Para hacer la representación gráfica de una función necesitamos un rango con los algunos pares ordenados (x_i, y_i) en el gráfico de la función. Luego estos pares se interpolan con un trazador cúbico (ver capítulo 2). Después de calcular este conjunto de pares ordenados, hay que hacer varias cosas. Primero necesitamos especificar la región dónde están los datos (en la hoja actual), en términos de filas y columnas: Fila de inicio, fila final, columna de inicio, columna final. Luego creamos un objeto gráfico, un rectángulo, especificando el largo, el ancho y la posición en la hoja en términos de su distancia al margen izquierdo y al margen superior de la hoja. En este rectángulo determina la posición de nuestro gráfico en la hoja. En este caso queremos que cada nuevo gráfico sustituya al anterior. Una vez que tenemos todos los elementos, incrustamos el "diagrama" e indicamos el tipo de gráfico (en nuestro caso `XYDiagram` con solo líneas). Finalmente hacemos algunos ajustes que tienen que ver tipo y tamaño de fuentes, color de fondo, etc.

Primero vamos a mostrar el código y luego se explica en detalle las partes de este código.Para la implementación vamos a usar como referencia la hoja de la figura (A.26),

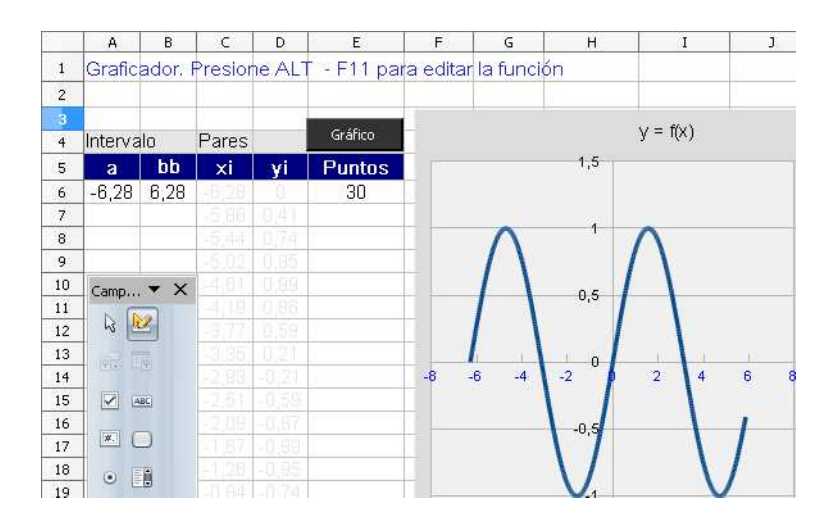


Figura A.26

El código completo de la subrutina `Graficar()` es,

Programa 50 *Subrutina para graficar*

```
Function f(x)
     f= x^3+x+1
End Function

Sub  Graficar(rango)
Dim Rango(0) As New com.sun.star.table.CellRangeAddress
Dim Rectangulo As New com.sun.star.awt.Rectangle
Dim Chart    As Object
Dim oChart   As Object
Dim Hoja
Dim a,b,xi, n

Hoja  = thisComponent.Sheets(0)
    a = Hoja.getCellRangeByName("A6").Value
    b = Hoja.getCellRangeByName("B6").Value
    n = Hoja.getCellRangeByName("E6").Value

    'Pares ordenados en las columnas C, D
      For  i=1 To n
        xi=a+(i-1)*(b-a)/n
        'Color de la fuente en celda: gris claro
        Hoja.getCellByPosition(2,4+i).CharColor = RGB( 230,230,230 )
        Hoja.getCellByPosition(3,4+i).CharColor = RGB( 230,230,230 )

        Hoja.getCellByPosition(2,4+i).Value=xi
        Hoja.getCellByPosition(3,4+i).Value=f(xi)
    next i
    'Insertar Gráfica
    'Medidas en centésimas de milímetro
 With Rectangulo
```

```
    .X = 6500       'Distancia desde la izquierda de la hoja
    .Y = 1000       'Distancia desde la parte superior
    .Width = 10000  'El ancho del gráfico
    .Height = 10000 'El alto del gráfico

                    'Anclaje de los datos
 With Rango(0)
    .Sheet =       0    'Hoja 1
    .StartColumn = 2    'C
    .EndColumn =   3    'D
    .StartRow =    5    'C6
    .EndRow =      5+n  'C[6+n]
 End With

Chart = thisComponent.Sheets(0).Charts
Chart.removeByName("Grafico")
Chart.addNewByName("Grafico",Rectangulo,Rango(),False, False)
oChart = Chart.getByName("Grafico").embeddedObject
oChart.diagram =oChart.createInstance("com.sun.star.chart.XYDiagram")
oChart.diagram.DataRowSource = com.sun.star.chart.ChartDataRowSource.COLUMNS

                                'Propiedades adicionales
 With oChart
      'Color de fondo, región externa.
      .getArea().FillBackground = True
      .getArea().FillStyle = RGB(220,220,220)
      .getArea().FillColor = RGB(220,220,220)
       ’Color de fondo, región interna
      .Diagram.getWall.FillBackground = True
      .Diagram.getWall.FillStyle = RGB(240,240,240)
      .Diagram.getWall.FillColor = RGB(240,240,240)

      .Diagram.SplineType = 1  'Interpola con trazadores cúbicos
      .Diagram.SymbolType =-1  'Símbolo para pares ordenados (none)
       'Ejes y textos
      .Diagram.HasXAxisTitle = True
      .Diagram.XAxisTiTle.string = "X"
      .Diagram.XAxisTiTle.CharHeight = 10
      .Diagram.XAxis.CharHeight = 8
      .Diagram.XAxis.Marks = 1
       '.Diagram.XAxis.StepMain = 1
      .Diagram.XAxis.StepHelp = 0
      .Diagram.XAxis.CharColor = RGB(0,0,255)
      .Diagram.XAxis.AutoStepMain = False
      .Diagram.XAxis.AutoStepHelp = False
      .Diagram.YAxis.CharHeight = 8
      .Diagram.YAxis.Origin=0
      '.Diagram.HasYAxisTitle = True
      '.Diagram.YAxisTiTle.string = ""
```

```
        '.Diagram.YAxisTiTle.CharHeight = 0
        .HasMainTitle = True
        .Title.string = "y = f(x)"
        .Title.CharHeight = 10
 End With
End Sub
```

Para calcular los pares ordenados necesitamos la función f, un intervalo $[a,b]$ y la cantidad n de puntos (por default será de 30). El código para el cálculo de los pares ordenados sería,

```
For i=1 To n
        xi = a +(i-1)*(b-a)/n
        Hoja.getCellByPosition(2,4+i).Value=xi
        Hoja.getCellByPosition(3,4+i).Value=f(xi)
Next i
```

Como son n nodos, dividimos el intervalo $[a,b]$ en n partes iguales, todas de tamaño $(b-a)/n$. Luego $x_0 = a$, $x_1 = a + (b-a)/n$, $x_2 = a + 2(b-a)/n, ..., x_n = b$.

Para el manejo del gráfico usamos cuatro objetos.

"`Rango()`" es un objeto que tiene soporte para usar los servicios del "servicio"[1] `com.sun.star.table.CellRangeAddress`, por tanto podremos tener acceso a la región de la hoja donde están los pares ordenados (estos serían los servicios del "servicio" `CellRangeAddress`),

```
Dim Rango(0) As New com.sun.star.table.CellRangeAddress
  ...
With Rango(0)
   .Sheet =       0    'Hoja 1
   .StartColumn = 2    'C
   .EndColumn =   3    'D
   .StartRow =    5    'C6
   .EndRow =      5+n  'C[6+n], después de n datos.
End With
```

El objeto `Rango` lo inicializamos en 0 si solo necesitamos ubicar una matriz de datos,en este caso los datos (x_i, y_i). Si necesitamos hacer referencia a $n+1$ bloques de datos en distintas partes (de la hoja o en otras hojas), ponemos `DimRango(n)` y el bloque de datos j se describe en términos de hoja, filas y columnas con `Rang(j).Sheet,...,Rango(j).EndRow`. Así, `Rango()` "tendrá" todos los datos.

Por ejemplo, si queremos la representación gráfica de dos funciones f y g en el mismo sistema y si, por alguna razón los datos $g(x_i)$ están algo lejos de las columnas `C,D` (en la misma hoja o otra hoja), entonces podríamos inicializar la matriz

[1]Los "servicios" son los componentes de OpenOffice. Un "servicio" es algo parecido a una clase o un "tipo" en Java. Las propiedades o métodos serían los servicios del "servicio". Ver [35].

de la variable `Rango` en 1; de esta manera, con `Rango(0)` indicamos la ubicación de los datos (x_i, y_i) y con `Rango(1)` la ubicación de los datos $g(x_i)$. Así, el objeto `Rango()` "tendrá" tres columnas y el tipo de gráfico que elegimos (de 'dispersión') representa los pares $(x_i, f(x_i))$ y los pares $(x_i, g(x_i))$. En nuestro chart aparecerán los dos gráficos.

"`Rectangulo`" es un objeto que tiene soporte para usar los servicios del "servicio" `com.sun.star.awt.Rectangle`, por tanto tenemos acceso a las propiedades (servicios) que nos permite dotar de dimensiones al rectángulo y posicionarlo en al hoja. Este rectángulo determina las dimensiones y la posición de nuestro gráfico.

```
Dim Rectangulo As New com.sun.star.awt.Rectangle
 ...
                   'Medidas en centésimas de milímetro
With Rectangulo
   .X = 6500       'Distancia desde la izquierda de la hoja
   .Y = 1000       'Distancia desde la parte superior
   .Width = 10000  'Ancho del rectángulo
   .Height = 10000 'Altura del rectángulo
End With
```

El objeto `Chart` tiene los "Charts" (gráficos) de la hoja. Usando `Chart` creamos la representación gráfica de los datos de `Rango(0)`. La posición de esta representación está determinada por `Rectangulo`. Para hacer todo esto se usa el método `addNewByName`,

```
Chart = thisComponent.Sheets(0).Charts
Chart.removeByName("Grafico")  'Remover gráfico anterior
Chart.addNewByName("Grafico",Rectangulo, Rango(),False, False)
```

Los parámetros `addNewByName` son: `"Nombre"`, `Rectangulo`, `Rango()`, `EncabezadoColumna` y `EncabezadoFila`.

`oChart` tiene acceso al gráfico, usando el nombre. Desde `oChart` especificamos algunas propiedades

```
oChart = Chart.getByName("Grafico").embeddedObject
oChart.Diagram =oChart.createInstance("com.sun.star.chart.XYDiagram")
oChart.Diagram.DataRowSource = com.sun.star.chart.ChartDataRowSource.COLUMNS
...
```

Ejercicios

A.1 Implementar una subrutina `Graficar(f,a,b)` que recibe una función `f` y la evalúa con la función *Eval* (de nuestra biblioteca `BblMatematica`) y hace la representación gráfica en $[a,b]$.

A.8 Modelo de Objetos de OOo.

Esta es una sección muy general que trata de dar una idea del manejo interno de OOo. En Java o VBA por ejemplo, uno puede crear clases, luego crear objetos para accesar las propiedades y métodos de la clase. En OOo Basic las cosas son algo diferentes. OOo Basic obtiene toda su funcionalidad de componentes "UNO" (Universal Network Objects, objetos de red universales). Un UNO esta compuesto de Servicios, Interfaces y Propiedades.
Un *servicio* es un componente de un UNO. Cada servicio consiste de una o más interfaces, las interfaces son conjuntos de métodos, para interactuar con los clientes (nuestros programas). Las propiedades son Constantes, Excepciones, Estructuras (strucs) y Enumeraciones.

Los servicios UNO están agrupados jerárquicamente en módulos, este capítulo todos los módulos que usamos están en un módulo central `com.sun.star`.

Por ejemplo, para accesar celdas de una hoja necesitamos el UNO `com.sun.star.table`. Este UNO tiene los servicios `Cell, CellCursor, CellRange, etc.`

Por ejemplo, el servicio `CellRange` tiene la interface `XCellRange` y esta interface tiene los métodos `getCellByPosition`, `getCellRangeByName` y `getCellRangeByPosition`.

Cuando en una macro necesitamos crear una instancia de un servicio, usamos la función `createUnoService()`. Una vez que referenciamos un servicio, podemos usar sus métodos y propiedades.

Por ejemplo, en nuestra `BblMatematica` instanciamos el servicio `FunctionAccess` del módulo `sheet` para accesar el método `callFunction`.

```
oFunction = createUnoService("com.sun.star.sheet.FunctionAccess")
cmax = oFunction.callFunction("MAX",oRange)
```

Todos los componentes de OpenOffice.org se pueden implementar en cualquier lenguaje que soporte UNO's. En C++ y Java se puede implementar componentes UNO. Cuando se instancia un UNO implementado en Java se levanta la máquina virtual de Java dentro de OpenOffice.org para atender la demanda.

Los componentes UNO *no* se pueden implementar con OOo Basic, solo los manejadores de eventos (Listeners). OpenOffice.org Basic es un lenguaje de scripts desarrollado para integrar directamente en OpenOffice.org pero no es la mejor opción para grandes proyectos aunque combina bien con otros lenguajes que requieren atajos rápidos y eficientes en ciertas tareas dentro de OOo.

Apéndice B

Conocimientos Previos

B.1 Inducción Matemática

Frecuentemente vamos a usar algunas propiedades de las funciones continuas, tanto a nivel teórico como a nivel de cálculo. Paralelamente se usan razonamientos que involucran *inducción matemática,* es decir, razonamientos acerca de afirmaciones que involucran un entero n y que para probarlos se necesita de un esquema de deducción que permita concluir que la afirmación es válida para todo entero n mayor que cierto entero inicial n_1.

Principio de Inducción Matemática.

Sea $A(n)$ una afirmación que contiene al entero n. Se puede concluir que la afirmación $A(n)$ es verdadera para toda $n \geq n_1$ si es posible

a.) probar que $A(n_1)$ es cierta

b.) probar que si se supone $A(k)$ verdadera para un k arbitrario pero $\geq n_1$, entonces $A(k+1)$ es verdadera.

*

■ **EJEMPLO B.1**

Afirmación $A(n):\ 1+r+r^2+\cdots+r^n=\dfrac{1-r^{n+1}}{1-r}$ si para toda $n\geq 0$, y r una constante diferente de 1.

¿Qué dice esta afirmación?. Que la suma de las primeras $n+1$ potencias de r se pueden calcular con una fórmula, sin necesidad de hacer la suma.

- Si $n=0,\ 1=\dfrac{1-r^{0+1}}{1-r}=\dfrac{1-r}{1-r}=1$
- Si $n=1,\ 1+r=\dfrac{1-r^2}{1-r}$
- Si $n=1000,\ 1+r+r^2+\cdots+r^{1000}=\dfrac{1-r^{1001}}{1-r}$
- Si $r=\dfrac{1}{2}$,

$$1+\frac{1}{2}+\left(\frac{1}{2}\right)^2+\cdots+\left(\frac{1}{2}\right)^{50}=\frac{1-\left(\frac{1}{2}\right)^{51}}{1-\frac{1}{2}}=2-\left(\frac{1}{2}\right)^{50}\approx 2.$$

Prueba de la afirmación $A(n)$.

Para probar que la afirmación es verdadera, usamos el principio de inducción

a.) Es cierta para $n=0$ pues $1=\dfrac{1-r^{0+1}}{1-r}=\dfrac{1-r}{1-r}$ si $r\neq 1$.

b.) *Suponemos* que es verdadera para $n=k$

$$1+r+r^2+\cdots+r^k=\frac{1-r^{k+1}}{1-r}\ \text{ si } r\neq 1$$

Ahora probamos que es cierta para $k+1$, es decir que

$$1+r+r^2+\cdots+r^k\ +\ r^{k+1}=\frac{1-r^{k+2}}{1-r}$$

En efecto

$$\begin{aligned}
\text{Como} \qquad\qquad 1+r+r^2+\cdots+r^k &= \frac{1-r^{k+1}}{1-r} \\
\text{entonces} \quad 1+r+r^2+\cdots+r^k\ +\ r^{k+1} &= \frac{1-r^{k+1}}{1-r}\ +\ r^{k+1} \\
&= \frac{1-r^{k+1}+r^{k+1}-r^{k+2}}{1-r} \\
&= \frac{1-r^{k+2}}{1-r}
\end{aligned}$$

EJERCICIOS

B.1 Verifique que $1+2+3\cdots+n=\dfrac{n(n+1)}{2}$ para toda $n\geq 1$

B.2 Verifique que $r^p+r^{p+1}+\cdots+r^{p+n}=\dfrac{r^p-r^{p+n+1}}{1-r}$ donde $r\neq 1$ y $p\in\mathbb{Z}^+$ son constantes.

B.3 Verifique que $(1+h)^n>1+nh$, para $n>1$ y h una constante no nula pero >-1.

B.2 Funciones continuas. Máximos y mínimos absolutos.

En todos los algoritmos de este capítulo se supone que se trabaja con funciones al menos continuas en un intervalo I, es decir funciones sin "huecos" ni asíntotas verticales en el intervalo. Esto es indispensable para que nuestros algoritmos de aproximaciones sucesivas no se vean detenidos por una situación de estas.

La continuidad es una propiedad "puntual". Una función es continua en todo un intervalo si es continua *en cada uno* de sus puntos.

Definición B.1 (Límites. Función continua.) *El símbolo* $\lim\limits_{x\longrightarrow a} f(x)=A$ *significa que para todo* $\epsilon>0$ *existe un* $\delta>0$ *tal que*

$$|f(x)\ -\ A|<\epsilon \ \ \textit{siempre que} \ \ 0<|x-a|<\delta$$

Una función $f:\mathbb{R}\longrightarrow\mathbb{R}$*, se dice continua en un punto* $x=a$ *si* $\lim\limits_{x\longrightarrow a} f(x)=f(a)$.

Una función $f:\mathbb{R}\longrightarrow\mathbb{R}$*, se dice continua en un conjunto* I *si* f *es continua en todos los puntos de* I

Las siguientes igualdades son equivalentes,

1. $\lim\limits_{x\longrightarrow a} f(x)=A$.

2. $\lim\limits_{x \longrightarrow a} f(x) - A = 0.$

3. $\lim\limits_{x \longrightarrow a} |f(x) \; - \; A| = 0.$

4. $\lim\limits_{h \longrightarrow 0} f(a+h) = A.$

■ **EJEMPLO B.2**

1. $y = \dfrac{\operatorname{sen}(x)}{x}$ no es continua en $x = 0$ aunque $\lim\limits_{x \longrightarrow 0} \dfrac{\operatorname{sen}(x)}{x} = 1$

2. $y = \dfrac{\operatorname{sen}(x)}{x}$ es continua en $[1, \frac{\pi}{2}]$

3. $y = \dfrac{\cos(x-1)}{x-1}$ no es continua en $x = 1$

Las representaciones gráficas se ven en la figura siguiente

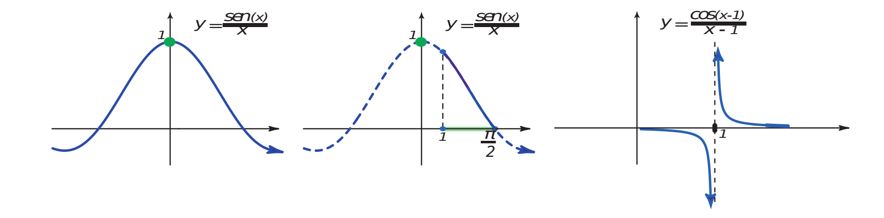


Figura B.1

Conocer la derivada de una función f sería suficiente para decidir si f es continua o no. Esto es así pues el cálculo de la derivada requiere la continuidad.

La continuidad y la derivabilidad en un punto requieren que la función este definida en un entorno abierto de éste. Como nos interesan los intervalos (ya sea abiertos o cerrados), definimos "continuidad por la derecha", "continuidad por la izquierda", "derivabilidad por la derecha" y "derivabilidad por la izquierda", usando límites unilaterales.

Cuando hablamos de continuidad o derivabilidad de una función en un intervalo cerrado, en los extremos del intervalo se debe entender que la derivabilidad o la continuidad es por "la derecha" o "por la izquierda".

Teorema B.1 *Si f es derivable en un intervalo I entonces f es continua en este intervalo.*

Por supuesto, el recíproco es falso. Una función puede ser continua en un punto pero no derivable. Tal es el caso de $f(x) = |x|$ que es continua en $x = 0$ pero no derivable (la gráfica presenta "un pico" en $x = 0$). Por esta razón en muchos teoremas se pide que una función sea "continua y derivable" aunque decir que es "derivable" sería suficiente.

■ **EJEMPLO B.3**

- La función $f(x) = \dfrac{1}{x-1} + \dfrac{1}{x-2} + \cdots + \dfrac{1}{x-3}$ es continua en $\mathbb{R} - \{1,2,3\}$ pues

$$f'(x) = \frac{-1}{(x-1)^2} - \frac{1}{(x-2)^2} - \frac{1}{(x-3)^2}$$

está definida en $\mathbb{R} - \{1,2,3\}$.

Máximos y mínimos absolutos.

Para analizar el error cometido en una aproximación del valor de una función f en un punto, la teoría establece, en general, el error exacto en términos de un valor $|g(\xi)|$ donde se sabe que ξ está en un intervalo I, pero es desconocido. En estos casos, la estimación del error requiere cambiar $|g(\xi)|$ por una cota superior de g en I. Usualmente la cota superior es el máximo absoluto de $|g|$ en I.

Definición B.2 (Máximos y mínimos absolutos.) *Sea f una función definida en un conjunto I de números reales. f tiene un máximo absoluto M en I si hay un punto $c_1 \in I$ tal que $f(x) \leq f(c_1)$, $\forall\, x \in I$. En este podemos poner $M = f(c_1)$.*

f tiene un mínimo absoluto m en I si hay un punto $c_2 \in I$ tal que $f(x) \geq f(c_2)$, $\forall\, x \in I$. En este podemos poner $m = f(c_2)$.

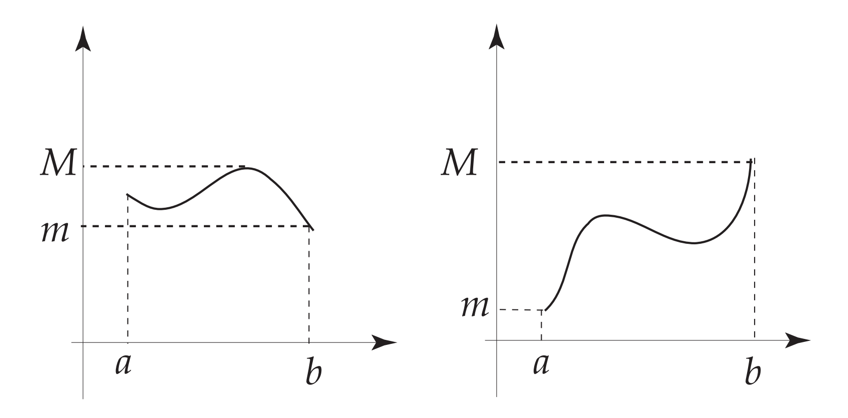


Figura B.2

Las funciones continuas tienen propiedades que requieren como hipótesis, que el intervalo donde estas propiedades se cumplen sea cerrado.

Teorema B.2 (de Weierstrass) *Si la función f es continua en el intervalo $[a,b]$ entonces f alcanza su máximo y su mínimo absoluto en* este intervalo

- Observe que el máximo absoluto de f, si existe, puede ser un *máximo relativo* o el valor de f en un extremo del intervalo I

- Observe que el mínimo absoluto de f, si existe, puede ser un *mínimo relativo* o el valor de f en un extremo de I

- Si $f'(x) \geq 0$ en $I = [a,b]$ entonces f es *creciente* en I. En este caso

$$m = f(a) \leq f(x) \leq f(b) = M$$

- Si $f'(x) \leq 0$ en $I = [a,b]$ entonces f es *decreciente* en I. En este caso

$$M = f(a) \geq f(x) \geq f(b) = m$$

- Si f es derivable en $I = [a,b]$, entonces f tiene extremos absolutos en I. Si $\{p_1,...p_n\}$ son los *puntos críticos*[1] de f en I, entonces

$$m = \text{Mín}\{f(a), f(p_1),..., f(p_n), f(b)\} \leq f(x) \leq M = \text{Máx}\{f(a), f(p_1),..., f(p_n), f(b)\}$$

- **Caso especial:** Sea f es derivable en $I = [a,b]$, sean $\{p_1,...p_n\}$ son los *puntos críticos* de f en I y $|A| = \{|f(a)|, |f(p_1)|,..., |f(p_n)|, |f(b)|\}$. Entonces:

  (a) Si f cambia de signo en I,

$$m = 0 \leq |f(x)| \leq M = \text{ Máx } |A|$$

  (b) Si f no cambia de signo en I,

$$m = \text{ Mín } |A| \leq |f(x)| \leq M = \text{ Máx } |A|$$

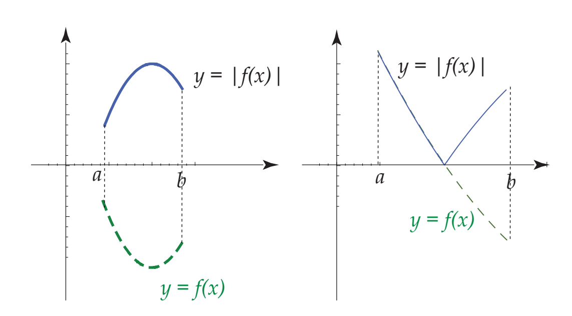


Figura B.3

■ EJEMPLO B.4

1. Sea $f(x) = 2x$. Sea $I = [1/2,4]$.

   Como $f'(x) = 2 > 0$ entonces f es creciente en I. Luego,

$$1 = f(1/2) \leq 2x \leq f(4) = 8$$

[1] p es punto crítico de f si $f'(p) = 0$ o si f' se indefine en p.

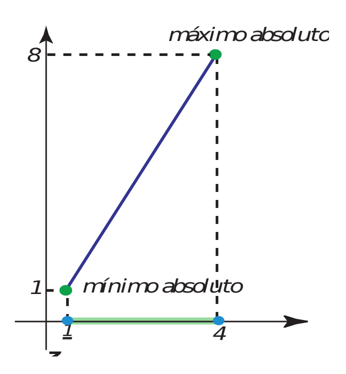


Figura B.4

2. Sea $f(x) = x - \cos(4x) - 1$. Sea $I = [1,2]$. En este caso la función no es monótona en I. Debemos comparar los valores de f en los puntos críticos en I con los valores de f en los extremos de I.

- puntos críticos

  $f'(x) = 1 + 4\sin(4x)$.

$$1 + 4\sin(4x) = 0 \implies \begin{cases} 4x = \operatorname{arcsen}(-1/4) + 2k\pi, \ k \in \mathbb{Z} \\ 4x = \pi + \operatorname{arcsen}(-1/4) + 2k\pi, \ k \in \mathbb{Z} \end{cases}$$

$$\implies \begin{cases} x = \dfrac{\operatorname{arcsen}(-1/4) + 2k\pi}{4}, \ k \in \mathbb{Z} \\ x = \dfrac{\pi - \operatorname{arcsen}(-1/4) + 2k\pi}{4}, \ k \in \mathbb{Z} \end{cases}$$

De todas las soluciones, la única que está en $I = [1,2]$ es

$$x = \frac{\operatorname{arcsen}(-1/4) + 2\pi}{4} \approx 1.50763$$

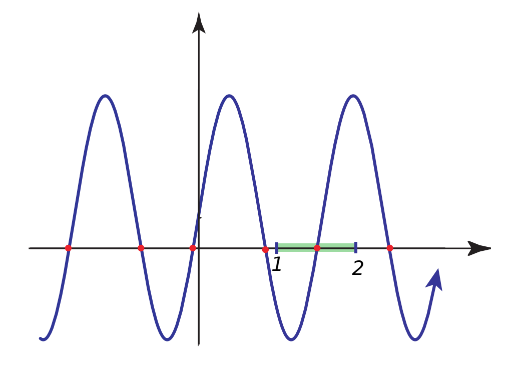


Figura B.5

- $m = \text{Mín}\{f(1), f(1.50763...), f(2)\} \approx \{0.6536, -0.4606, 1.1455\}$
- $M = \text{Máx}\{f(1), f(1.50763...), f(2)\}$

$m \approx -0.4606$

$M \approx 1.1455$

Así, $-0.4606... \leq x - \cos(4x) - 1 \leq 1.1455...$ en $I = [1,2]$

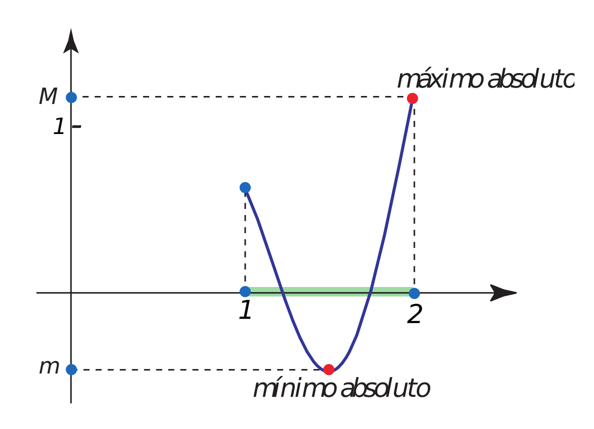


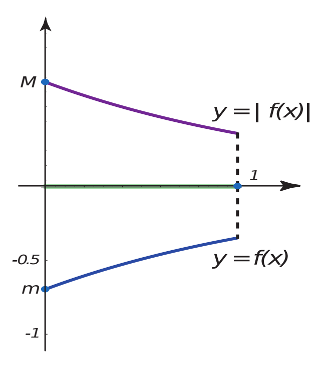
Figura B.6

Una teorema que nos será muy útil es

Teorema B.3 (Acotación para funciones continuas) *Sea f continua en un intervalo $[a,b]$. Entonces existe un número $K \geq 0$ tal que $|f(x)| \leq K$ para todo $x \in [a,b]$*

■ EJEMPLO B.5

- Sea $f(x) = -2^{-x}\ln(2)$. Si $I = [0,1]$,



$|f(x)| \leq M = |f(0)| \approx 0.693147$

La existencia de soluciones de la ecuación $f(x) = 0$ esta basado en el siguiente teorema

Teorema B.4 (Teorema del valor Intermedio) *Si f es continua en $[a,b]$ con $f(a) \neq f(b)$, entonces para cada $c \in [f(a), f(b)]$ existe una preimagen $x = p$ en $[a,b]$ tal que $f(p) = c$.*

Corolario B.1 *Si f es continua en $[a,b]$ y si $f(a)$ y $f(b)$ difieren en el signo, existe $p \in]a,b[$ tal que $f(p) = 0$.*

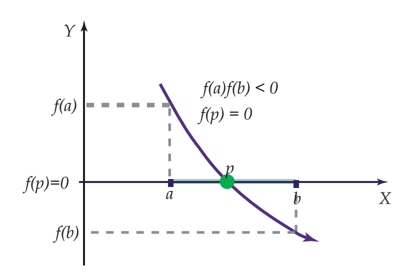


Figura B.7

■ **EJEMPLO B.6**

1. La función $f(x) = x^3 + x + 1$ tiene un *cero* en $[-1,0]$ pues $f(-1)f(0) = -1 < 0$

2. La función $f(x) = \dfrac{1}{x-1} + \dfrac{1}{x-2} + \cdots + \dfrac{1}{x-3}$ tiene un *cero* en $]1,2[$ y otro *cero* en $]2,3[$ pues, evaluando en $x = 1.2,\ x = 1.8,\ x = 2.8$ podemos verificar un cambio de signo.

$$f(1.2) = 3.194...,\ f(1.8) = -4.58333...,\ f(2.2) = 4.58333...,\ f(2.8) = -3.19444...$$

Además son los únicos *ceros*, ya que f es decreciente:

$$f'(x) = \frac{-1}{(x-1)^2} - \frac{1}{(x-2)^2} - \frac{1}{(x-3)^2} < 0$$

Otro teorema que nos será útil es

Teorema B.5 *Si f es continua en $x = p$ y si $f(p) \neq 0$, existe entonces un intervalo $]p-\delta, p+\delta[$ en el que f tiene el mismo signo que $f(p)$.*

Figura B.8

EJERCICIOS

B.4 Calcule el máximo y mínimo absoluto de las siguientes funciones, en el intervalo que se indica

a) $f(x) = e^x$ en $[0,2]$
b) $f(x) = 80x^3 + 12\,\text{sen}(2x)$ en $[1,2]$
c) $f(x) = |80x^3 + 12\,\text{sen}(2x)|$ en $[-2,1]$
d) $f(x) = \frac{1}{2}(\cos(x) + \text{sen}(x))$ en $[-1,1]$.
e) $f(x) = \dfrac{-2+4x^2}{e^{x^2}}$ en $[-1,1]$.
f) $f(x) = \dfrac{1}{x-1} + \dfrac{1}{x-2} + \cdots + \dfrac{1}{x-3}$ en $[6/5, 9/5]$
g) $f(x) = |(x^2-1)/3|$ en $[-0.5,0]$

B.5 Considere $g(x) = 2^{-x}$.
a) Muestre que $g(x) \in [1/3,1]$ si $x \in [1/3,1]$
b) ¿Podría encontrar una constante positiva $k < 1$ tal que $|g'(x)| \leq k \;\; \forall x \in\,]1/3,1[$?

B.6 Sea $g(x) = (x^3-1)/4$.
a) Muestre que $g(x) \in [-1,1]$ si $x \in [-1,1]$
b) ¿Podría encontrar una constante positiva $k < 1$ tal que $|g'(x)| \leq k \;\; \forall x \in [-1,1]$?

B.7 Verifique que $f(x) = 1 + 4\text{sen}(4x)$ tiene un cero en $[1,2]$.

B.8 Verifique que $f(x) = x^3 + x + 1$ tiene solamente un cero en $\mathbb{R}$.

B.9 Si $f(x) = -\tan(x + \pi/10)$, determine los extremos absolutos de $|f'(x)|$ en $[-1,1]$

B.10 Considere $g(x) = 1/2\,(\text{sen}(x) + \cos(x))$. Calculando el máximo y el mínimo absoluto de g en $[0,1]$, verifique que

$$0.270151... \leq g(x) \leq 0.920735... \quad \text{en} \quad [0,1].$$

$$-0.150584... \leq g'(x) \leq 0.5 \quad \text{en} \quad]0,1[.$$

B.11 Sea $g(x) = (x^2-1)/3$. Verifique que

$$0 \leq g(x) \leq -1/3 \quad \text{si} \quad x \in [-1,1]$$

$$|g'(x)| \leq 2/3 \quad \text{si} \quad x \in [-1,1]$$

B.12 Sea $f(x) = \ln x$. Verifique que si $\xi \in [1,e]$,

$$\left|\frac{f''(\xi)}{2}(x-1)(x-e)\right| \leq \frac{1}{2}|(x-1)(x-e)|$$

B.13 Considere ET_n, ES_n y EG_n definidas de la siguiente manera:

$$ET_n = \frac{(b-a)^3}{12n^2}\cdot f''(\xi), \quad \text{con} \quad \xi \in]a,b[,$$

$$ES_n = \frac{(b-a)^5}{180n^4} f^{(4)}(\xi), \quad \text{con} \quad \xi \in]a,b[,$$

$$EG_n = \frac{2^{2n+1}[(n)!]^4}{(2n+1)[(2n)!]^3} g^{(2n)}(\xi) \quad \text{con} \quad \xi \in]-1,1[\quad \text{y } g(x) = \frac{b-a}{2} f\left(\frac{a+b+(b-a)x}{2}\right).$$

a) Si $f(x) = \operatorname{sen} x$ y $a = 0$, $b = 1$, $n = 10$; estimar $|ET_{10}|$ y $|ES_{10}|$

b) Si $f(x) = e^{-x}$ y $a = 0$, $b = 1$, $n = 3$; estimar $|EG_3|$

B.3 Teorema de Taylor

El teorema de Taylor no dice que si tenemos dos números α y u en un intervalo I, y si una función f es $n+1$ veces derivable en I, entonces $f(\alpha)$ se puede aproximar con un polinomio $P_n(x)$ de grado n para el que $P_n(u) = 0$. Además podemos estimar el error cometido usando una cota superior de $f^{(n+1)}$ entre α y u. Adicionalmente, si α es desconocida y se sabe que $f(\alpha) = 0$ entonces podemos obtener una aproximación de α.

Teorema B.6 (Teorema de Taylor con forma diferencial del resto) *Supongamos que f es una función con derivadas continuas hasta el orden $n+1$ en un intervalo I Dados $\alpha, u \in I$, existe ξ, en el intervalo cerrado con extremos u y α, tal que*

$$f(\alpha) = f(u) + f'(u)(\alpha-u) + \frac{f''(u)}{2!}(\alpha-u)^2 + \cdots + \frac{f^{(n)}(u)}{n!}(\alpha-u)^n + \frac{f^{(n+1)}(\xi)}{(n+1)!}(\alpha-u)^{n+1}$$

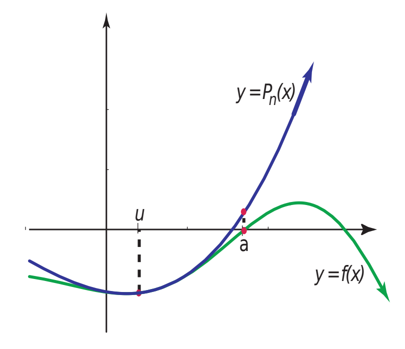


Figura B.9

- $P_n(x) = f(u) + f'(u)(x-u) + \dfrac{f''(u)}{2!}(x-u)^2 + \cdots + \dfrac{f^{(n)}(u)}{n!}(x-u)^n$ es el *polinomio de Taylor* de orden n alrededor de $x = u$.

- $\dfrac{f^{(n+1)}(\xi)}{(n+1)!}(\alpha - u)^{(n+1)}$ es el *resto*.

- Se puede escribir $f(\alpha) \approx P_n(\alpha)$ con error $\dfrac{f^{(n+1)}(\xi)}{(n+1)!}(\alpha-u)^{n+1}$, con ξ entre u y α.

- Como en general ξ es un número desconocido, es útil observar que si $|f^{(n+1)}(x)| \leq M$ en un intervalo que contenga a α y a u, entonces podemos obtener *una estimación del resto*

$$\left|\frac{f^{(n+1)}(\xi)}{(n+1)!}(\alpha-u)^{n+1}\right| \leq \left|\frac{M}{(n+1)!}(\alpha-u)^{n+1}\right|$$

  Observe que si u está suficientemente cercano a α o si n es suficientemente grande, entonces el *resto* podría ser despreciable.

■ **EJEMPLO B.7**

1. Como $f(x) = e^x$ y todas sus derivadas son continuas en todo $\mathbb{R}$, entonces podemos aproximar e^2 con un polinomio d Taylor de grado 10 alrededor de $u = 0$. En este caso, $\alpha = 2$ y

$$\begin{aligned} e^2 &= f(0) + f'(0)(2-0) + \frac{f''(0)}{2!}(2-0)^2 + \cdots + \frac{f^{(n)}(0)}{n!}(2-0)^n + \frac{f^{(n+1)}(\xi)}{(n+1)!}(2-0)^{n+1} \\ &= 1 + 2 + \frac{2^2}{2} + \cdots + \frac{2^n}{n!} + \frac{f^{(n+1)}(\xi)}{(n+1)!}\,2^{n+1} \quad \text{con } \xi \text{ entre } 0 \text{ y } 2 \\ &\leq 1 + 2 + \frac{2^2}{2} + \cdots + \frac{2^n}{n!} + \frac{e^2 2^{n+1}}{(n+1)!} \quad \text{pues } f^{(n+1)}(u) = e^u \leq e^2 \text{ en } [0,2]. \end{aligned}$$

   En particular, $e^2 \approx 1 + 2 + \dfrac{2^2}{2} + \cdots + \dfrac{2^{10}}{10!} = 7.3889...$ con error $\leq \dfrac{e^2 2^{11}}{(11)!} = 0.000379...$

2. Sea $Q(x) = a_0 + a_1 x + \cdots + a_n x^n$ con $a_n \neq 0$. Como $Q^{(n+1)}(x) \equiv 0$ entonces el polinomio de Taylor $P_n(x)$ es idéntico a $Q(x)$ (pues el resto es 0).

3. Podemos aproximar $\cos(1)$ con el polinomio de Taylor para $\cos(x)$, de orden $n = 3$, alrededor de $u = \pi/2 \approx 1.57079$.

   Como $P_3(x) = -(x - \pi/2) + \frac{1}{6}(x - \pi/2)^3$ entonces $\cos(1) \approx P_3(1) = 0.5398...$ con error

$$\left|\frac{\cos(\xi)(1-\pi/2)^4}{4!}\right| \leq \left|\frac{1\cdot(1-\pi/2)^4}{4!}\right| = 0.00442...$$

pues $|\cos(x)| \leq 1\ \forall\, x.$

4. Si α es un *cero* de $f(x) = 1+4\operatorname{sen}(4x)$ en $[1,2]$ entonces si $u \in [1,2]$, podemos escribir

$$f(\alpha) = 0 = f(u) + f'(u)(\alpha-u) + \frac{f''(\xi)}{2!}(\alpha-u)^2$$

de donde, despejando α (del factor lineal) obtenemos

$$\alpha = u - \frac{f(u)}{f'(u)} - \frac{f''(\xi)}{2f'(u)}(\alpha-u)^2$$

Si el *resto* es despreciable (es decir, si u está suficientemente cercano a α), entonces

$$\alpha \approx u - \frac{f(u)}{f'(u)}$$

- Por ejemplo, si $u = 1.5$ entonces $\alpha \approx 1.50765892743399...$

Un caso especial del teorema de Taylor el el teorema del valor medio para derivadas

Teorema B.7 (Teorema del valor medio para derivadas) *Si f es continua y derivable en $[a,b]$ entonces existe $\xi \in]a,b[$ tal que*

$$\frac{f(b)-f(a)}{b-a} = f'(\xi)$$

EJERCICIOS

B.14 Calcule n tal que $P_n(1)$ aproxime $\cos(1)$ con un error estimado ≤ 0.0000005.

B.15 Calcule el Polinomio de Taylor $P_3(x)$, alrededor de $u=0$, para $Q(x) = x^3 - x + 1$

B.16 Calcule el Polinomio de Taylor $P_3(x)$, alrededor de $u=\pi/2$, para $Q(x) = x^3 - x + 1$ y verifique que $P_3(x) = Q(x)$

B.17 Aproxime el único cero de $f(x) = x^3 + x + 1$ en $[-1,0]$, usando $P_2(x)$ alrededor de $u=-0.5$ (la solución con doce decimales exactos es $-0.682327803828...$).

B.4 Notación O de Landau

Es muy adecuado recurrir a algunas funciones cuyo comportamiento es muy familiar, para compararlas con expresiones más complejas en un entorno de x_0 o en el infinito. Nos interesa saber si una función f es tan "rápida" como g analizando si el cociente $\left|\frac{f}{g}\right|$ permanece acotado en un entorno dado.

Definición B.3 *Decimos que* $f \in O(g)$ *conforme* $x \longrightarrow \infty$ *si existe* M *y* $K > 0$ *tal que*

$$|f(x)| \leq K|g(x)| \text{ para todo } x > M.$$

$f \in O(g)$ *conforme* $x \longrightarrow x_0$ *si existe* $\delta > 0$ *y* $K > 0$ *tal que*

$$|f(x)| \leq K|g(x)| \text{ con } |x - x_0| < \delta$$

$f \in o(g)$ *conforme* $x \longrightarrow x_0$, *si*

$$\lim_{x \longrightarrow x_0} \frac{f(x)}{g(x)} = 0$$

EJEMPLO B.8

1. $x_n = \dfrac{n^2 - 1}{n^3} \in O(1/n)$ conforme $n \longrightarrow \infty$ ya que $\dfrac{n^2 - 1}{n^3} \leq \dfrac{1}{n}$ para todo $n > 1$.

2. Si $f \in o(1)$ entonces $\lim_{x \longrightarrow x_0} f(x) = 0$

   Abusando del lenguaje escribimos $\lim_{x \longrightarrow x_0} o(1) = 0$

3. Como $\dfrac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1} \in o\left((x - x_0)^n\right)$ cuando $x \longrightarrow x_0$, se puede escribir

$$f(x) = f(x_0) + f'(u)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n + o((x - x_0)^n)$$

4. Como $\dfrac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1} \in O\left((x - x_0)^{n+1}\right)$ cuando $x \longrightarrow x_0$, se puede escribir

$$f(x) = f(x_0) + f'(u)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n + O((x - x_0)^{n+1})$$

5. Algunos *desarrollos limitados* conforme $x \longrightarrow 0$

   (a) $\ln(x+1) = x - x^2/2 + o(x^2)$

   (b) $\ln(x+1) = x + o(x)$

   (c) $\tan(x) = x + o(x^2)$

   (d) $\tan(x) = x + o(x\ \ )$ (solo cambia de $o(x^2)$ a $o(x)$)

   (e) $\text{sen}(x) = x + o(x)$

   (f) $x = x + o(x)$ (igual que la de seno!)

B.4.1 Propiedades de $o(g)$. Cálculo de límites.

El teorema de Taylor nos permite aproximar funciones complicadas con polinomios. Puesto que es fácil operar con polinomios, esto puede ayudar a simplificar algunas expresiones complicadas que aparecen en el cálculo de límites, por ejemplo. Solo necesitamos conocer la manera de operar con expresiones 'del tipo $o(g)$'.

Propiedades de $o(g)$

- $o(g) \pm o(g) = o(g)$
- $k \cdot o(g) = o(g)$, $k \neq 0$.
- $f \cdot o(g) = o(fg)$
- $o(g)^n = o(g^n)$

Usando estas propiedades podemos calcular límites operando con expansiones de Taylor. Primero se hace una simplificación de una parte de la expresión, usando desarrollos limitados, y luego para las otras partes de la expresión se busca un desarrollo limitado adecuado para obtener la simplificación de la expresión. Observe que para una función dada, se pueden usar distintos desarrollos limitados.

EJEMPLO B.9

1. Vamos a calcular $\displaystyle\lim_{x\longrightarrow 0} \frac{\tan(x) - \ln(x+1)}{\tan(x)\ln(x+1)}$ usando los desarrollos limitados adecuados.

$$
\begin{aligned}
\lim_{x\longrightarrow 0}\frac{\tan(x)-\ln(x+1)}{\tan(x)\ln(x+1)} &= \lim_{x\longrightarrow 0}\frac{x+o(x^2)-(x-x^2/2+o(x^2))}{\tan(x)\ln(x+1)}\\
&= \lim_{x\longrightarrow 0}\frac{x^2/2+o(x^2)}{(x+o(x))(x+o(x))}\\
&= \lim_{x\longrightarrow 0}\frac{x^2/2+o(x^2)}{x^2(1+o(1))(1+o(1))}\\
&= \lim_{x\longrightarrow 0}\frac{1/2+o(1)}{(1+o(1))(1+o(1))}=1/2
\end{aligned}
$$

2. $\underline{\lim_{x\longrightarrow 0}\frac{\text{sen}(x)}{x}} = \lim_{x\longrightarrow 0}\frac{x+o(x)}{x+o(x)}=1$

EJERCICIOS

Calcular los siguientes límites usando desarrollos limitados

B.18 $\lim_{x\longrightarrow 0}\frac{x}{\text{sen}(x)}$

B.19 $\lim_{x\longrightarrow 0}\frac{\ln(x+1)}{x+\tan(x)}$

B.5 Sucesiones

Definición B.4 *Una sucesión* $x0, x_1, ..., x_k, \cdots$, *denotada* $\{x_n\}_{n=0}^{\infty}$ *o simplemente* x_n, *es una función* $x:\mathbb{N}\longrightarrow\mathbb{R}$.

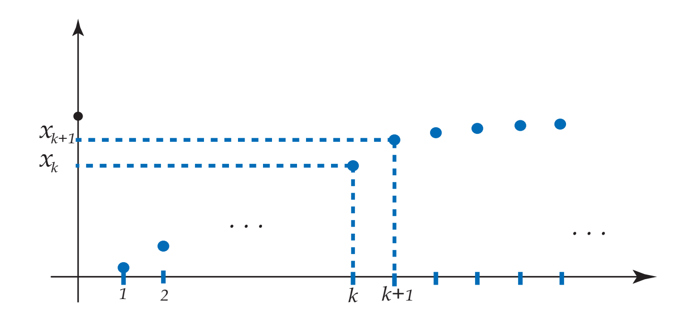


Figura B.10

Notación: Se acostumbra escribir "x_n" en vez de la notación funcional $x(n)$ y, abusando del lenguaje, se dice "la sucesión x_n".

Una sucesión puede ser definida por una *relación de recurrencia* cuando se conocen uno o varios valores iniciales y una relación entre x_n y uno o más términos anteriores.

■ **EJEMPLO B.10**

1. $x_n = \dfrac{(-1)^n + 1}{2}, \; n \geq 0.$

   En este caso los primeros términos de la sucesión son

$$1, 0, 1, 0, 1, 0, 1, \cdots$$

2. **(Relación de recurrencia).** $x_0 = 1, \; x_n = 0.5\left(x_{n-1} + \dfrac{A}{x_{n-1}}\right), \; A \geq 0.$

   En este caso, si $A = 2$, los primeros términos de la sucesión (usando 5 dígitos de precisión) son

$$1, 1.5000, 1.4167, 1.4142, 1.4142, 1.4142, \cdots$$

3. **(Relación de recurrencia).** $x_0 = 1, \; x_n = x_{n-1} - \dfrac{f(x_{n-1})}{f'(x_{n-1})}$

   En este caso, si $f(x) = x^2 - 2$, los primeros términos de la sucesión (usando 5 dígitos de precisión) son

$$1, 1.5000, 1.4167, 1.4142, 1.4142, 1.4142, \cdots$$

EJERCICIOS

B.20 Dé el término general x_n de la sucesión $-1, 1, -1, 1, -1, ...$

B.21 Verifique que si $f(x) = x^2 - A$, la sucesión definida por

$$x_0 = 1, \; x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$$

es equivalente a $x_0 = 1, \; x_n = 0.5\left(x_{n-1} + \dfrac{A}{x_{n-1}}\right)$.

B.22 Sea $x_n = \dfrac{x_{n-1}}{2} + \dfrac{1}{2}$. Verifique, usando inducción matemática, que $x_n = \dfrac{x_0}{2^n} + \displaystyle\sum_{k=1}^{n} \dfrac{1}{2^k}$

Definición B.5 (Límite) *Decimos que la sucesión x_n tiene límite L, lo que se escribe*

$$\lim_{n \to \infty} x_n = L$$

si para cualquier $\epsilon > 0$ existe un número natural N tal que si $n \geq N$ entonces $|x_n - L| < \epsilon$

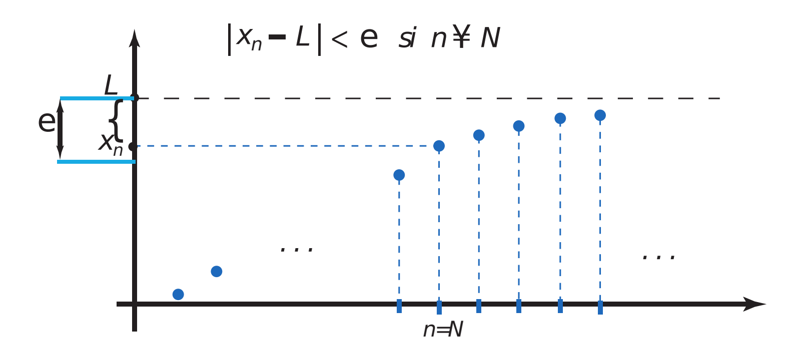


Figura B.11

Como muchas veces tratamos con valores absolutos y con términos de sucesiones tales como x_{n+k}, el siguiente teorema es muy útil

Teorema B.8 *Si* $\lim_{n \to \infty} |x_n - L| = 0$ *entonces* $\lim_{n \to \infty} x_n = L$

Si k es un entero positivo y si $\lim_{n \to \infty} |x_n - L| = 0$ *entonces* $\lim_{n \to \infty} x_{n+k} = L$

El cálculo del límite de una sucesión (si existe), se puede hacer usando las mismas técnicas usadas para el cálculo del límite de una función.

Teorema B.9 (Cálculo de Límites) *Si* $\lim_{x \to \infty} f(x) = L$ *y si f está definida en $\mathbb{Z}^+$, entonces*

$$\lim_{n \to \infty} f(n) = L$$

Las sucesiones convergentes permanecen acotadas pero las sucesiones acotadas no siempre convergen, solo las sucesiones monótonas (siempre crecientes o siempre decrecientes) acotadas, convergen.
Además, si una sucesión converge, la distancia entre sus términos se hace pequeña, tanto como se quiera, a partir de

un subíndice N.

Teorema B.10 *Si la sucesión x_n es una sucesión monótona acotada, x_n converge.*
Si la sucesión x_n es convergente, x_n es acotada.
Si la sucesión x_n es convergente, dado cualquier $\epsilon > 0$, existe un natural N tal que $|x_n - x_{n+1}| < \epsilon$ para todo $n \geq N$.

■ **EJEMPLO B.11**

Sea $x_0 = 0$ y $x_{n+1} = \sqrt{x_n + 2}$. La sucesión x_n converge por ser monótona y acotada. Veamos

i. x_n es creciente, es decir $x_{n+1} \geq x_n$ para toda n. En efecto, procediendo por inducción

(a) $x_1 = \sqrt{2} \geq x_0$

(b) Supongamos que $x_n \geq x_{n-1}$

$$\begin{aligned} x_n &\geq x_{n-1} \\ x_n + 2 &\geq x_{n-1} + 2 \\ \sqrt{x_n + 2} &\geq \sqrt{x_{n-1} + 2}, \text{ pues } y = \sqrt{x} \text{ es creciente.} \\ x_{n+1} &\geq x_n \end{aligned}$$

ii. Usando inducción podemos verificar x_n está acotada por 2, es decir $x_n \leq 2$ para toda n. En efecto,

(a) $x_0 \leq 2$

(b) Supongamos que $x_n \leq 2$

$$\begin{aligned} x_n &\leq 2 \\ x_n + 2 &\leq 4 \\ \sqrt{x_n + 2} &\leq \sqrt{4} \\ x_{n+1} &\leq 2 \end{aligned}$$

■ **EJEMPLO B.12**

1. $\lim_{n\to\infty} (1/2)^n = 0$ pues $\lim_{x\to\infty} (1/2)^x = 0$.

2. Si $x_0 = 1$ y $x_{n+1} = \frac{1}{2} x_n$ entonces $\lim_{n\to\infty} x_n = 0$ pues

$$x_{n+1} = \frac{1}{2} x_n = \left(\frac{1}{2}\right)^2 x_{n-1} = \cdots = \left(\frac{1}{2}\right)^{n+1} x_0$$

3. A veces debemos usar la definición de límite de una sucesión para comprobar algún resultado.

   Esta definición dice que el límite de la sucesión es L si la distancia entre x_n y L (o sea $|x_n - L|$) se puede hacer tan pequeña como queramos a partir de un subíndice k en adelante.

   Podemos mostrar que $\lim\limits_{n\to\infty}(1/2)^n = 0$ usando la definición, de la siguiente manera:

   - Habría que verificar que dado un número $\epsilon > 0$ tan pequeño como queramos, siempre es posible encontrar un subíndice k a partir del cual $|x_k - 0| < \epsilon$. En este caso el razonamiento es sencillo, a partir de la desigualdad $|x_k - 0| < \epsilon$, buscamos el "k" a partir del cual esta afirmación se cumple.

$$\begin{aligned} & |x_k - 0| &<& \epsilon \\ \Longleftrightarrow\; & |(1/2)^k - 0| &<& \epsilon \\ \Longleftrightarrow\; & (1/2)^k &<& \epsilon, \text{ pues } (1/2)^k > 0 \\ \Longleftrightarrow\; & k\ln(1/2) &<& \ln(\epsilon) \\ \Longleftrightarrow\; & k &>& \frac{\ln(\epsilon)}{\ln(1/2)}, \text{ pues } \ln(1/2) < 0 \end{aligned}$$

   Esto demuestra que siempre hay un subíndice k a partir del cual la distancia de $(1/2)^k$ a 0 es tan pequeña como queramos.

   Por ejemplo si $\epsilon = 1.0 \times 10^{-17}$ entonces $|(1/2)^k - 0| < 1.0 \times 10^{-17}$ si

$$k > \frac{\ln(1.0 \times 10^{-17})}{\ln(1/2)} \approx 56.47$$

El siguiente teorema, conocido de manera coloquial como *teorema del sándwich* o *teorema del emparedado,* es muy útil para decidir algunos resultados sobre sucesiones.

Teorema B.11 (Teorema de intercalación) *Si existe un entero $N > 0$ tal que $a_n \leq x_n \leq b_n$, $\forall\, n > N$, y si el límite de las sucesiones a_n y b_n existe cuando $n \longrightarrow \infty$, entonces si*

$$\lim_{n\longrightarrow\infty} a_n = L = \lim_{n\longrightarrow\infty} b_n$$

también $\lim\limits_{n\longrightarrow\infty} x_n = L$

El teorema dice no solo que el límite de x_n existe, también que es igual a L. En los siguientes ejemplos vamos a ver una aplicaciones de carácter teórico que aparecen en algunos razonamientos que están mas adelante.

■ **EJEMPLO B.13**

1. Si $|x_n - L| \le K\left(\frac{1}{2}\right)^n$, $\forall\, n \ge N$ entonces

$$0 \le |x_n - L| \le K\left(\frac{1}{2}\right)^n$$

es decir, $|x_n - L|$ está entre las sucesiones $a_n = 0$ y $b_n = K\left(\frac{1}{2}\right)^n$, $\forall\, n \ge N$. Ahora, como $\lim\limits_{n\longrightarrow\infty} 0 = 0 = \lim\limits_{n\longrightarrow\infty} K\left(\frac{1}{2}\right)^n$, se concluye

$$\lim_{n\longrightarrow\infty} |x_n - L| = 0 \text{ o también } \lim_{n\longrightarrow\infty} x_n = L$$

2. Supongamos que $|x_n - L| \le \frac{M}{n}$, $\forall\, n \ge N$ (con $N \in \mathbb{N}, M \in \mathbb{R}$) entonces

$$0 \le |x_n - L| \le \frac{M}{n}$$

es decir, $|x_n - L|$ está entre las sucesiones $a_n = 0$ y $b_n = \frac{M}{n}$, $\forall\, n \ge N$. Ahora, como $\lim\limits_{n\longrightarrow\infty} 0 = 0 = \lim\limits_{n\longrightarrow\infty} \frac{M}{n}$, se concluye

$$\lim_{n\longrightarrow\infty} |x_n - L| = 0 \text{ o también } \lim_{n\longrightarrow\infty} x_n = L$$

EJERCICIOS

B.23 Calcule $\lim\limits_{n\longrightarrow\infty} \frac{\cos(n)}{n}$

B.24 Verifique que $\lim\limits_{n\longrightarrow\infty} \left(1 + \frac{1}{n}\right)^n = e$

B.25 Si $0 < \alpha < 1$, $e_0 > 0$ y $|e_n| < \alpha^n e_0$, calcule $\lim\limits_{n\longrightarrow\infty} |e_n|$

B.26 Suponga que $0 < \alpha < 1$, $e_0 > 0$ y $|e_{n+1}| < \alpha\, e_n$ para toda $n > 1$. Verifique que $\lim\limits_{n\longrightarrow\infty} |e_n| = 0$

B.27 Calcule $\lim\limits_{n\longrightarrow\infty} x_n$ si $x_n = \frac{x_{n-1}}{2} + \frac{1}{2}$.

B.28 $(\star)$ Sea $x_0 = 0$ y $x_{n+1} = \sqrt{x_n + 2}$. Verifique que la sucesión converge y que $\lim\limits_{n\longrightarrow\infty} x_n = 2$.

B.29 $(\star)$ Verifique que $\lim\limits_{n\longrightarrow\infty} x_n = 2$ si $x_1 = \sqrt{2}$ y $x_n = \sqrt{2x_{n-1}}$.

B.30 Verifique que la sucesión $a_1=\sqrt{2}$, $a_2=\sqrt{2}^{\sqrt{2}}$, $a_3=\sqrt{2}^{\sqrt{2}^{\sqrt{2}}}$,..., es convergente.

Solución: Procedemos por inducción. Observe que $a_n=\sqrt{2}^{\,a_{n-1}}$, $n=2,3,...$ entonces como la exponencial a^x $(a>1)$ es creciente, se tiene

i.) a_n es creciente. En efecto, $a_2>a_1$ pues $\sqrt{2}^{\sqrt{2}}>\sqrt{2}$. Además si $a_n>a_{n-1}$ entonces $a_{n+1}=\sqrt{2}^{\,a_n}>\sqrt{2}^{\,a_{n-1}}=a_n$

ii.) $a_n<2$, $n=1,2,\ldots$. En efecto, $a_1=\sqrt{2}<2$. Además si $a_n<2$ entonces $a_{n+1}=\sqrt{2}^{\,a_n}<\sqrt{2}^{\,2}=2$.

B.6 Teorema del valor medio para integrales

Si en $[x_i,x_{i+1}]$ G es continua y φ integrable y de un mismo signo, entonces existe $\varepsilon_i\in]x_i,x_{i+1}[$ tal que $\int_{x_i}^{x_{i+1}}G(x)\varphi(x)\,dx=G(\varepsilon_i)\int_a^b\varphi(x)\,dx$.

■ **EJEMPLO B.14**

$(x-a)(x-b)$ es integrable (y de primitiva conocida) y de un mismo signo en $[a,b]$ (negativa), entonces:

$\int_a^b(x-a)(x-b)e^{x^3}dx=-e^{\varepsilon^3}\,\frac{(b-a)^3}{6}$ con $\varepsilon\in]a,b[$.

Apéndice C

Bits y Bytes

La memoria del computador está formada por una gran cantidad de bytes y cada byte está constituido por 8 bits. Un bit puede almacenar un 1 o un 0.
Un *bit* (binary digit) se refiere a un dígito en el sistema numérico en base 2. Por ejemplo $(10010111)_2$ es un número de 8 bits de largo
Un *byte* es una colección de bits. Actualmente un byte son 8 bits.
Una *palabra* (word) es un grupo de bits. El tamaño de palabra típico puede ser 32 o 64 bits, etc.
Cuando se habla de capacidad de almacenamiento, se acostumbra llamar kilobit (kb) a $2^{10} = 1024$ bits, megabit (mb) a $2^{20} = 1024$ kb, gigabit (gb) a 1024 mb y terabit (tb) a 2^{40} bits.

*

Ejemplo C.1

- Una variable entera (`int`) está formada por 4 bytes, es decir 32 bits. Estos 32 bits representan el valor almacenado por esa variable en binario.
- Si una variable tipo `int` almacena el número $5 = 1 * 2^2 + 0 * 2^1 + 1 * 2^0$, su representación en memoria sería algo como

$$\underbrace{0000\,0000}_{\text{1er byte}}\;\underbrace{0000\,0000}_{\text{2do byte}}\;\underbrace{0000\,0000}_{\text{3er byte}}\;\underbrace{0000\,0101}_{\text{4to byte}}$$

- Como tenemos 32 bytes, las variables `int` solo pueden almacenar enteros entre $10\ldots0000000 = -2^{31}$ (el -0 no cuenta) y $2^{31} - 1 = (01\ldots1111111)_2$
- Las variables `long` usan 64 bits y almacenan enteros entre -2^{63} y $2^{63} - 1$.

Apéndice D

¿Porqué $1.0000000... = 0.999999...$?

Tal vez, la mejor manera de tratar este detalle es indicar que es un convenio. Aún así se puede verificar su consistencia con la teoría de varias maneras, unas más rigurosas que otras.
Los números racionales son insuficientes para hacer mediciones en el mundo real. Un ejemplo clásico es el triángulo rectángulo con catetos de medida 1. La hipotenusa medirá $\sqrt{2}$, que no es un número racional. Los números reales son una *completación* de los números racionales. Muchos números que usamos de manera natural como $e, e^{1.4142}, e^{\sqrt{2}}$, etc., solo tienen significado como límites de una sucesión de números racionales. Por ejemplo, $\sqrt{2}$ es el límite de la sucesión de números racionales

$$1,\ 1.4,\ 1.41,\ 1.414,\ 1.4142,\ ...$$

Los números reales pueden ser definidos (construidos) como límites de sucesiones (de Cauchy) de números racionales. Una sucesión x_n se dice que es una sucesión de Cauchy si para cualquier $\epsilon > 0$ siempre se puede encontrar un N a partir del cual $|x_n - x_m| < \epsilon$ para todo $n, m > N$ (o sea, los términos de la sucesión están tan cerca como se quiera a partir de algún N). Dos sucesiones de Cauchy x_n, y_n son *equivalentes* si $(x_n - y_n)$ tiene límite 0. Esto define una *relación de equivalencia*. Una sucesión x_n representa a todas las sucesiones equivalentes a ella. En particular, el número real 0.99999... es el límite de la sucesión (de Cauchy)

$$0,\ 0.9,\ 0.99,\ 0.999,\ 0.9999,\ ...$$

que es *equivalente* a la sucesión de Cauchy

$$1,\ 1.0,\ 1.00,\ 1.000,\ 1.0000,\ ...$$

*

que tiene como límite el número real 1. Como ambas sucesiones son equivalentes, representan el mismo número real. O sea, $0.9999... = 1$.
Como un ejemplo de consistencia, observe que como $ar + ar^2 + \cdots = \dfrac{ar}{1-r}$ si $|r| < 1$, entonces

$$0.9999... = 9\left(\frac{1}{10}\right) + 9\left(\frac{1}{10}\right)^2 + 9\left(\frac{1}{10}\right)^3 \cdots = \frac{9\left(\frac{1}{10}\right)}{1 - \frac{1}{10}} = 1$$

Apéndice E

Programación Visual Basic (VBA) para Excel

Prof. Walter Mora F.,
Prof. José Luis Espinoza B.
Escuela de Matemática
Instituto Tecnológico de Costa Rica.

E.1 Introducción

Microsof Excel© es un software para el manejo de hojas electrónicas agrupadas en *libros* para cálculos de casi cualquier índole. Entre muchas otras aplicaciones, Excel se utiliza para modelar un sinnúmero de aplicaciones a la Ingeniería, en el tratamiento estadístico de datos, así como para la presentación gráfica de los mismos. La hoja electrónica Excel es ampliamente conocida por profesionales y estudiantes en proceso de formación, pero hay una gran cantidad de usuarios que no conocen a profundidad su gran potencial y adaptabilidad a l0 .os diferentes campos del conocimiento.

Para científicos e ingenieros, el Excel constituye una herramienta computacional muy poderosa. También tiene gran utilidad en la enseñanza de las ciencias y la Ingeniería, particularmente, en la enseñanza de los métodos numéricos. Pese a que existen en el mercado programas computacionales muy sofisticados, tales como MATLAB, MATHEMATICA, etc., no están tan disponibles como Excel, que usualmente forma parte del paquete básico de software instalado en las computadoras que funcionan bajo el sistema Windows© de Microsoft.

*

A continuación se brinda al lector una breve introducción a algunas actividades de programación con macros escritos en VBA (una adaptación de Visual Basic para Office de Microsoft), definidos desde una hoja electrónica de Excel. Salvo pequeñas diferencias, el material puede ser desarrollado en cualquier versión.

La hoja electrónica servirá tanto como medio de entrada entrada de datos a los programas, como de salida o despliegue de resultados.

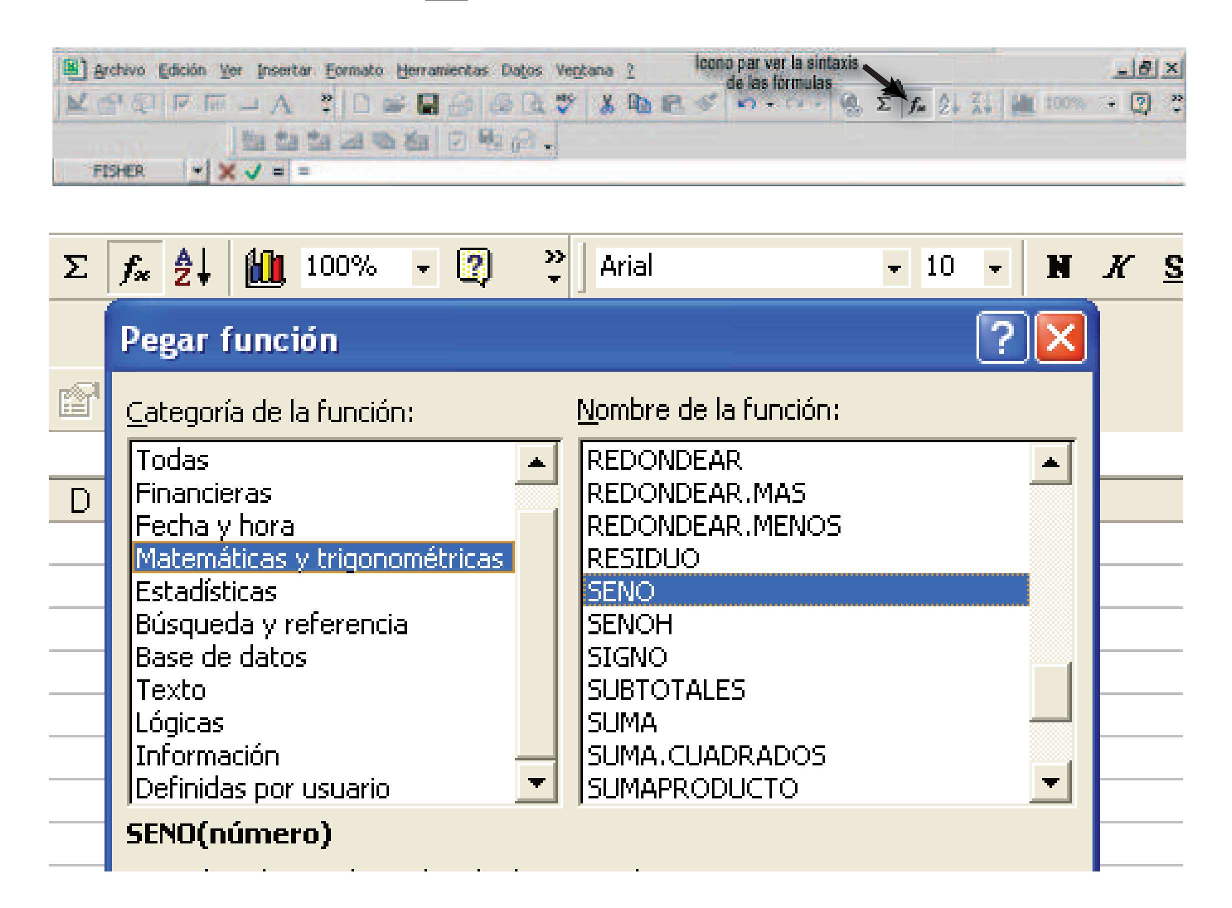
E.2 Evaluación de funciones

E.2.1 Funciones definidas por el usuario

A manera de ejemplo, vamos a evaluar la función

$$f(x) = 2x^3 + \ln(x) - \frac{\cos(x)}{e^x} + \text{sen}(x)$$

1. Como al evaluar $f(x)$ se debe recurrir a varias funciones básicas que se invocan desde Excel, se puede tener acceso a su sintaxis, pulsando el ícono $\boxed{f_x}$ y seleccionar "Matemáticas y Trigonométricas".



2. Para escribir una fórmula, seleccionamos una celda para escribir el valor a ser evaluado; por ejemplo, podemos digitar el valor 1.1 en la celda B3.

3. Ahora en la celda C3 digitamos, de acuerdo a la sintaxis de una versión de Excel en español[1], la fórmula:

```
=2*B3^3+LN(B3)-COS(B3)/EXP(B3)+SENO(B3)
```

Una vez que ha sido digitada, simplemente se pulsa la tecla "Entrar" o "Enter".

E.2.2 Errores comunes

Conforme vamos digitando nuestras primeras fórmulas, nos van apareciendo algunos errores que usualmente son debidos a un manejo inadecuado de la sintaxis o a la incompatibilidad con la configuración de la computadora. A continuación se describen algunas situaciones que pueden aparecer.

1. El valor de error **#¿NOMBRE?** aparece cuando Excel no reconoce texto en una fórmula. Debe revisar la sintaxis de dicha fórmula o, si es una macro, verificar que esté en un módulo de esta hoja.

2. El valor de error **#¡VALOR!** da cuando se utiliza un tipo de argumento (u operando) incorrecto. Este error se da por ejemplo, cuando evaluamos una función numérica en una celda que contiene algo que no sea un número (Por defecto, el contenido de una celda vacía es cero).

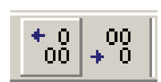
3. El valor de error **#¡NUM!** se aparece cuando hay un problema con algún número en una fórmula o función. Por ejemplo, si evaluamos una función logarítmica en cero o en un número negativo.

4. El valor de error **#¡DIV/0!** se produce cuando en una fórmula se divide por 0 (cero).

5. El valor de error **#¡REF!** se da cuando una referencia a una celda no es válida.

6. Dependiendo de la forma en que esté configurado el sistema Windows, debe usarse punto o coma para separar la parte decimal de los números a evaluar. Para personalizarlo, se debe entrar al panel de control y en la "Configuración regional" se selecciona "Números". En la primera cejilla, "Símbolo Decimal" se selecciona el punto o la coma, según sea el caso. Finalmente, se presiona el botón "Aplicar" y luego "Aceptar".

7. Una situación que a veces es confundida con un error se da cuando el sistema trabaja con poca precisión y se presentan valores numéricos no esperados. Por ejemplo, si el formato de una celda se ha definido para dos posiciones, entonces la operación `+1.999+1` efectuado en dicha celda dará como resultado el valor **2**, que no es otra cosa que el resultado de tal suma redondeado a dos decimales. El valor correcto se obtiene aumentado la precisión con el ícono correspondiente:



[1]En una versión en español se usa SENO(x) para evaluar la función *Sen(x)*, mientras que una versión en inglés se usa SIN(x).

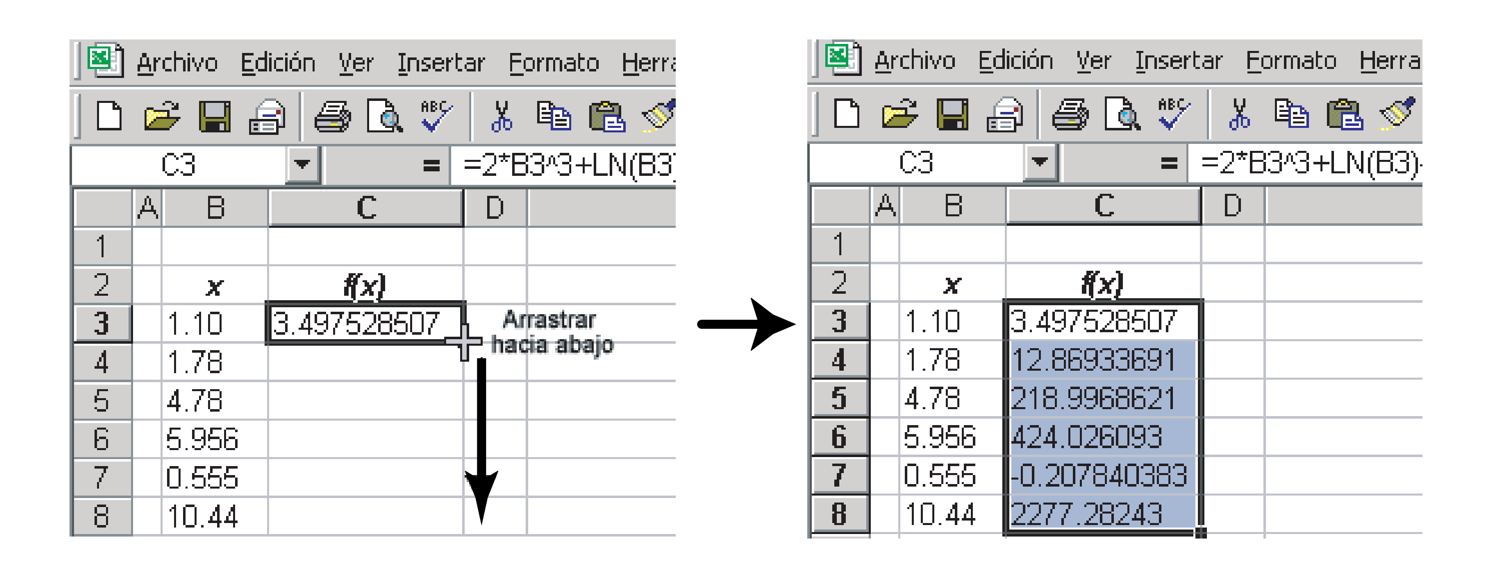
También se puede cambiar la precisión en el menú **"Formato–Celdas–Número–Posiciones decimales"**.

Estos cambios son sólo de apariencia, pues, independientemente del número de dígitos que sean desplegados, Excel manipula los números con una precisión de hasta 15 dígitos. Si un número contiene más de 15 dígitos significativos, Excel convertirá los dígitos adicionales en ceros (0).

E.2.3 Evaluando una función en varios tipos de parámetros

Muchas fórmulas a evaluar tienen argumentos de distinto tipo, pues algunos argumentos varían (a veces con un incremento determinado), mientras que otros permanecen constantes. Por lo general estos argumentos son tomados de celdas específicas, por lo que es importante saber manejar distintos escenarios para la evaluación de una función o fórmula.

E.2.3.1 Evaluación con argumentos variables Continuando con el ejemplo que iniciamos en la sección 1.2.1, a partir de la celda B4 podemos continuar digitando valores, siempre en la columna B y con el cuidado de que estos números no se salgan del dominio de la función $f(x) = 2x^3 + \ln(x) - \dfrac{\cos(x)}{e^x} + \operatorname{sen}(x)$, que en este caso es el conjunto de los números reales positivos. Una vez hecho ésto, se evalúa la función $f(x)$ en la celda C3, como se hizo previamente. Luego, seleccionamos esta misma celda C3 y se ubica el *mouse* en la esquina inferior derecha, arrastrándolo hasta la celda deseada. Otra posibilidad es hacer un doble **`clic`** en la esquina inferior derecha de la celda a copiar y esto realiza la copia automáticamente.



E.2.3.2 Evaluación con argumentos variables y/o constantes Es común tener que evaluar funciones o fórmulas que dependen de varios parámetros, algunos de los cuales se mantienen fijos mientras que otros son variables.

Ejemplo E.1

El siguiente ejemplo describe una función con dos parámetros y una variable.

La función[a] $P(t) = \dfrac{K}{1+Ae^{-kt}}$, con $A = \dfrac{K-P_0}{P_0}$, describe el tamaño de una población en el momento t.
Aquí:

- k es una constante de proporcionalidad que se determina experimentalmente, dependiendo de la población particular que está siendo modelada,
- P_0 es la población inicial y
- K es una constante llamada *capacidad de contención* o *capacidad máxima que el medio es capaz de sostener*.

Si queremos evaluar $P(t)$ para distintos valores del tiempo t en días, seguimos la siguiente secuencia de pasos:

1. Para empezar, es importante escribir encabezados en cada una de las columnas (o filas) donde vamos a escribir los datos que serán los argumentos de la función. En este caso, comenzando en la celda B3, escribimos las etiquetas

   `P0    K    k   t    P(t)`.

2. A continuación escribimos los valores de los parámetros, comenzando en la celda B4

   `100   1000  0.08  0`.

3. Ahora escribimos la fórmula de la función $P(t)$ en la celda G4:

   ```
   =C$4/(1+((C$4-B$4)/B$4)*EXP(-D$4*E4))
   ```

   Como puede observarse, el único argumento variable es t y nos interesa mantener a los otros argumentos constantes. Para mantener un valor (que se lea en una celda) constante, se le agrega el símbolo $ antes del número de fila, como por ejemplo `C$4`.

   En nuestro ejemplo, los argumentos constantes son los que están en las celdas B4, C4 y D4, mientras que el valor de t en la celda E4, es variable.

4. Finalmente, escribimos varios valores para t en la columna E, seleccionamos la celda F4 y arrastramos para evaluar $P(t)$ en el resto de valores de t.

	A	B	C	D	E	F
1						
2						
3		P0	K	k	t	P(t)
4		100	1000	0,08	0	100
5					5	142,18925
6					6	152,229
7					10	198,2569
8					15	269,48745
9					17	302,11964
10					20	354,97892
11					21	373,50145
12					30	550,52086
13					45	802,62394

[a] $P(t)$ es la solución de la llamada *ecuación logística:* $\dfrac{dP}{dt} = kP\left(1-\dfrac{P}{K}\right)$

E.2.3.3 Construyendo rangos con un incremento fijo A menudo necesitamos evaluar una función en una secuencia de valores igualmente espaciados, por lo que a continuación se explica cómo hacerlo, modificando el ejemplo previo de crecimiento de una población.

1. En la celda C5 podemos escribir un valor, por ejemplo, 5, que servirá como incremento entre un tiempo y el siguiente, iniciando con $t = 0$.

2. En la celda E4 escribimos el tiempo inicial $t = 0$ y en la celda E5 se escribe el nuevo tiempo con el incremento h:

   ```
   =+E4+C$6
   ```

   En esta operación debemos usar el símbolo `$` frente al número de fila en `C$6` para que el incremento se mantenga inalterado al copiar esta operación en otra celda situada en una fila diferente. De igual forma, si necesitáramos copiar la expresión en otra celda situada en diferente columna, se debe escribir el símbolo `$` frente al número de columna.

3. Ahora seleccionamos esta celda `E5` y la arrastramos hacia abajo para obtener los nuevos tiempos con el respectivo incremento.

	A	B	C	D	E	F
1						
2						
3		P0	K	k	t	P(t)
4		100	1000	0,1	0	100
5					5	142,1893
6	**Incremento:**	*h*=	5		10	198,2569
7					15	269,4875
8					20	354,9789
9					25	450,8531
10					30	550,5209
11					35	646,291
12					40	731,6039
13					45	802,6239

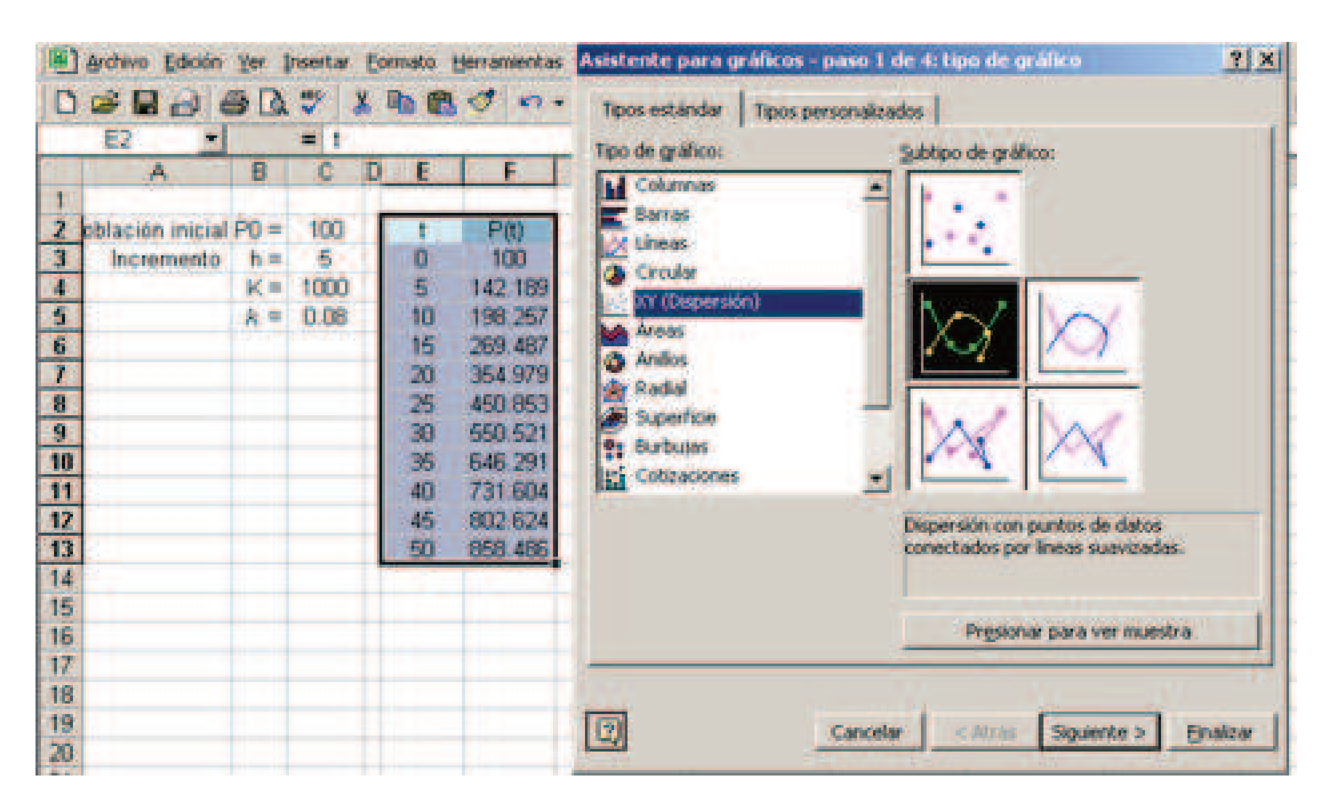
Nota: Esto también se puede hacer escribiendo, en celdas consecutivas, un valor y luego el valor más el incremento y luego seleccionando ambas celdas para luego arrastrar. Sin embargo, al leer el incremento de una celda, se actualizan todos los datos que hacen referencia a ésta, por lo que es más práctico y flexible.

E.3 Gráficas

Continuando con el ejemplo anterior en el que se ha evaluado la población $P(t)$ en un conjunto de valores del tiempo t, recordemos que en las columnas E y F se han escrito, respectivamente, los valores de t y $P(t)$. Para graficar $P(t)$ con respecto a t, podemos seguir los siguientes pasos:

1. Seleccionamos el rango en el cual se encuentran los valores de t y $P(t)$. Este rango puede incluir las celdas que contienen los rótulos de las columnas.

2. Presionamos el icono, que activa el asistente para gráficos.
   Hay varias opciones que podemos elegir para el gráfico y en nuestro caso podemos elegir el tipo **Dispersión.**



3. Presionamos el botón Siguiente y luego Finalizar. Antes de finalizar se pueden escoger distintas opciones para personalizar el gráfico. A continuación se muestra la salida del gráfico.

	A	B	C	D	E	F
1						
2	Población inicial	P0 =	100		t	P(t)
3	Incremento	h =	5		0	100
4		K =	1000		5	142.19
5		k =	0.08		10	198.26
6					15	269.49
7					20	354.98
8					25	450.85
9					30	550.52
10					35	646.29
11					40	731.6
12					45	802.62
13					50	858.49

La curva obtenida se llama *curva logística* o *sigmoide*, por la forma de *"S"* que tiene.

E.4 Programación de macros

E.4.1 Introducción

El lenguaje Visual Basic para Aplicaciones (VBA), en el contexto de Excel, constituye una herramienta de programación que nos permite usar código Visual Basic adaptado para interactuar con las múltiples facetas de Excel y personalizar las aplicaciones que hagamos en esta hoja electrónica.

Las unidades de código VBA se llaman **`macros`**. Las macros pueden ser *procedimientos* de dos tipos:

- Funciones (**Function**)

- Subrutinas (**Sub**)).

Las funciones pueden aceptar argumentos, como constantes, variables o expresiones. Están restringidas a entregar un valor en una celda de la hoja o en el contexto de otra función o subrutina. Las funciones pueden llamar a otras funciones y hasta subrutinas (en el caso de que no afecten la entrega de un valor en una sola celda).

Una subrutina realiza acciones específicas pero no están destinadas a devolver un solo valor. Puede aceptar argumentos, como constantes, variables o expresiones y puede llamar funciones que ejecutan ciertas acciones.

Con las subrutinas podemos entregar valores en distintas celdas de la hoja. Es ideal para leer parámetros en algunas celdas y escribir en otras para completar un cuadro de información a partir de los datos leídos.

E.4.1.1 Editar y ejecutar macros.

Las funciones y las subrutinas se pueden implementar en el editor de Visual Basic (Alt-F11).

Para usar una función en una hoja de Excel se debe, en el editor de VB, insertar un módulo y editar la función en este módulo. Esta acción de describe más adelante. De la misma manera se pueden editar subrutinas en un módulo.

Una función definida por el usuario se invoca en una hoja de la misma forma en que se invocan las funciones predeterminadas de Excel o una fórmula. Una subrutina se puede invocar por ejemplo desde la ventana de ejecución de macros (Alt-F8) o desde un botón que hace una llamada a la subrutina (como respuesta al evento de hacer **`clic`** sobre él, por ejemplo).

El código que ejecuta un botón puede llamar a subrutinas y a las funciones de la hoja. El código del botón no está en un módulo. En la hoja de edición donde se encuentra el código del botón, se pueden implementar funciones para uso de este código pero que serán desconocidas para la hoja (mensaje de error **`#¿NOMBRE?`**).

Nota: un error frecuente es editar una función en un módulo que corresponde a una hoja y llamarlo desde otra hoja. En este caso se despliega el error (mensaje de error **`#¿NOMBRE?`**).

E.4.2 Funciones

Una función tiene la siguiente sintaxis:

Function NombreFun(*arg1, arg2,...,argn*)

Declaración de Variables y constantes

Instruccción 1

Instruccción 2

...

Instruccción k

NombreFun = *Valor de retorno*

'comentario

End Function

Una función puede tener o no tener argumentos, pero es conveniente que retorne un valor. Observe que se debe usar el nombre de la función para especificar la salida:

NombreFun = *Valor de retorno*

Nota 1: Al interior de las funciones o subrutinas, se pueden hacer comentarios utilizando el apóstrofo 'o también se usa **rem** antes de éstos. El uso de comentarios ayuda a documentar los programas.

Nota 2: Para el uso de nombres de variables o de cualquier otra palabra reservada de VBA, no se discrimina entre el uso de letras mayúsculas y minúsculas.

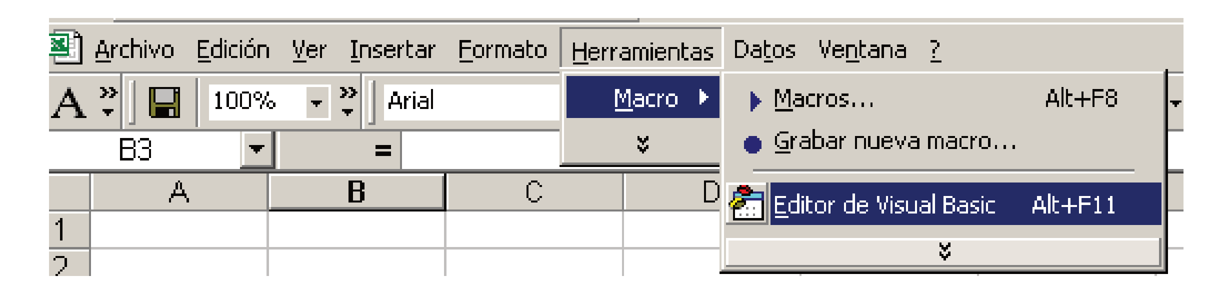
Nota 3: No debemos usar nombres para las funciones como f1(x), f2(x) o g1(x), etc. pues a al hora de llamarlas desde Excel, se confunden con las celdas F1, G1, etc. y como resultado se produce un error.

Implementar una función. Vamos a implementar como una macro la función con la que se trabajó previamente:

$$f(x) = 2x^3 + \ln(x) - \frac{\cos(x)}{e^x} + \operatorname{sen}(x).$$

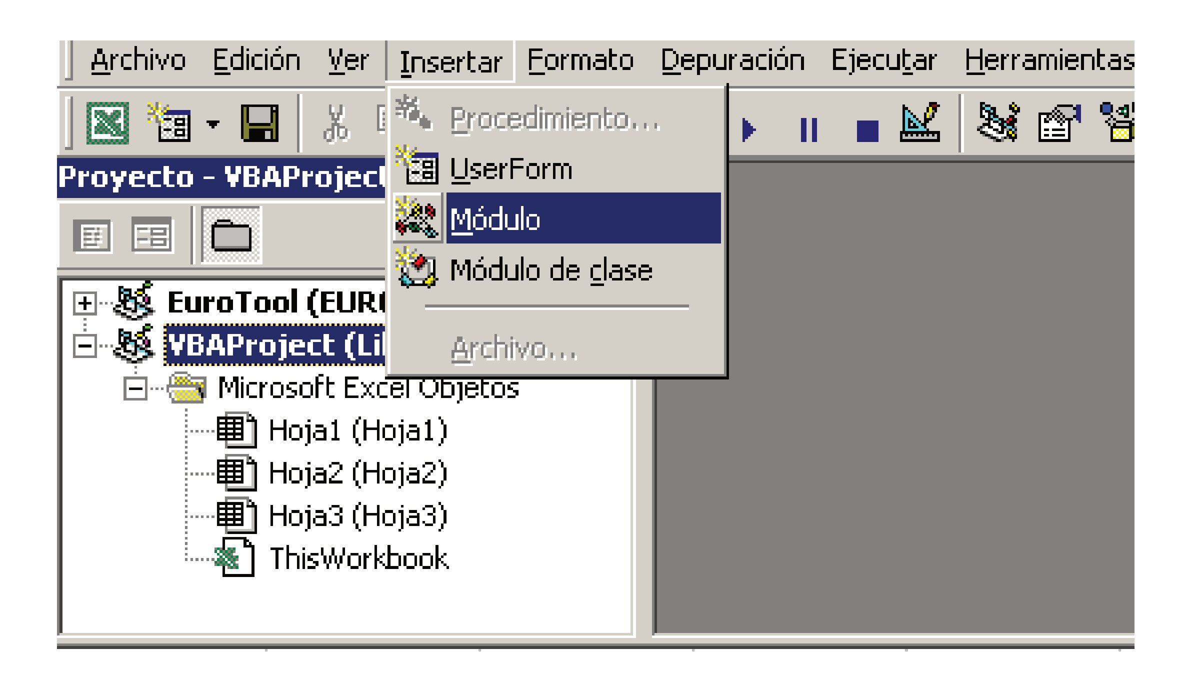
Para su definición y utilización, se siguen los pasos:

1. Ingresamos al menú y en la opción **Herramientas** seleccionamos **Macros**. Luego se elige **Editor de Visual Basic**:



También puede usar **Alt-F11**

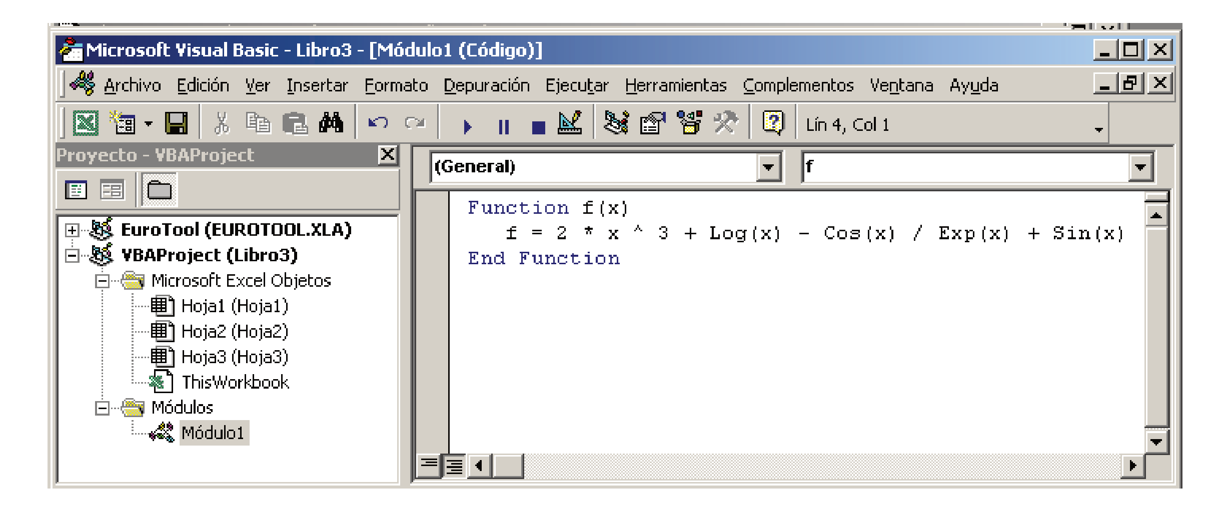
2. Nuevamente, en el menú de la ventana que se abre, se elige **Insertar**, para luego seleccionar **Módulo**:



3. Ahora en la pantalla de edición del módulo, escribimos el siguiente código:

Código VBA E.1:

```
Function f(x)
  f = 2 * x ^ 3 + Log(x) - Cos(x) / Exp(x) + Sin(x)
  rem retorna la imagen de f(x)
End Function
```



4. Una vez que ha sido editado el código de la macro, se salva y salimos del ambiente de programación en Visual Basic para volver a la hoja electrónica de donde partimos. Esto se hace en el menú **Archivo**, seleccionando **Cerrar y Volver a Excel**.

5. Para evaluar la función $f(x)$ en algunos valores que se localicen, por ejemplo, desde la fila 3 hasta la fila 6 de la columna B, nos situamos en una celda en la que nos interese dejar el resultado de dicha evaluación y se digita `+f(B3)`. Luego se arrastra hasta C6 para copiar la fórmula, quedando:

	A	B	C	D
1				
2		x	f(x)	
3		1.5	8.13717649	
4		2.4		
5		5.6		
6		7.8		

	A	B	C	D
1				
2		x	f(x)	
3		1.5	8.13717649	
4		2.4	29.2658268	
5		5.6	352.320632	
6		7.8	952.156645	

E.4.3 Funciones en VBA y Funciones en Excel.

Hay ciertos detalles que debemos tomar en cuenta cuando trabajamos con VBA y Excel.

- En Excel **-3^2=9** aunque en VBA **-3^2=-9**.

  Lo mejor es siempre poner paréntesis **-(3^2)=-9** o **(-3)^2)** para no hacer la diferencia de si está en VBA o Excel.

- En la tabla que sigue hay una pequeña lista de funciones y su respectivo código en VBA y en Excel

```
VBA                 Excel
--------------------------------------------------
sgn(x)              signo(x)
sqr(x)              raiz(x)  (o sqrt(x) en inglés)
Log(x)              Ln(x)
Log(x)/Log(10)      Log(x)
...
```

Por ejemplo, consideremos la función

$$f(x) = 2x^3 + \ln(x) - \frac{\cos(x)}{e^x} + \text{sen}(x):$$

- En Excel la sintaxis es: `2*B3^3+LN(B3)-COS(B3)/EXP(B3)+SENO(B3)`
- En VBA la sintaxis es: `2 * x ^ 3 + Log(x) - Cos(x) / Exp(x) + Sin(x)`

- Para conocer con detalle la sintaxis de las funciones matemáticas estándar que se pueden evaluar en Visual Basic, puede usarse la **`Ayuda`** del **`Editor de Visual Basic`**.

Lectura de parámetros en celdas. Una vez más vamos a trabajar con el modelo de crecimiento poblacional descrito anteriormente. La función

$$P(t) = \frac{K}{1 + Ae^{-kt}},$$

con

$$A = \frac{K - P_0}{P_0}.$$

Ahora evaluaremos $P(t)$ para distintos valores del tiempo t en días, pero esta vez haremos dicha evaluación mediante una macro para definir $P(t)$.

Los parámetros los vamos a leer desde unas celdas ubicadas en la columna C. Para hacer referencia a una celda, se usa el código

`Cells(fila,columna)`

pero escribiendo "columna" en formato numérico. Por ejemplo, la celda C5 se invoca como

`Cells(5,3)`

Lo primero que hacemos es escribir, en el editor de VBA, la fórmula de $P(t)$, luego la invocamos en la celda F3 (de nuestra hoja de ejemplo) y arrastramos. Para ésto, se siguen los siguientes pasos:

1. En primer lugar, abrimos una hoja Excel, que se llame por ejemplo **`Poblacion.xls`**. Luego se escriben los valores de los parámetros, tal y como puede observarse en la siguiente figura:

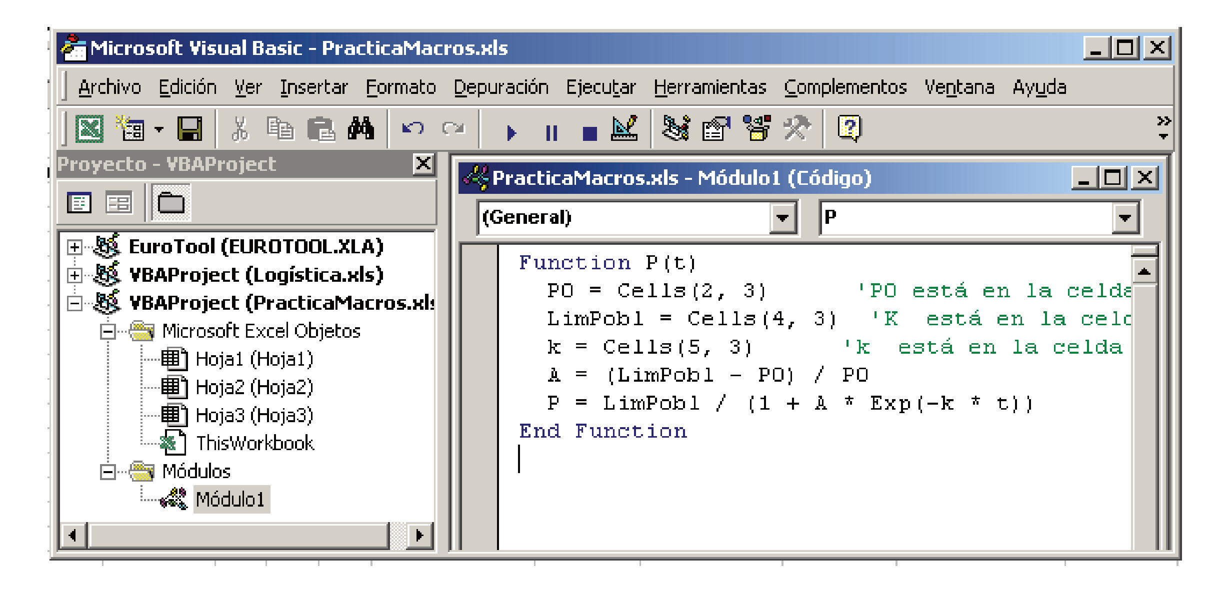
	A	B	C	D	E	F	G	H	I	J	K
1											
2	Población inicial	P0 =	100		t	0	5	10	15	20	
3	Incremento	h =	5		P(t)						
4		K =	1000								
5		k =	0.08								
6											

2. Ahora ingresamos al menú y en la opción **`Herramientas`** seleccionamos **`Macros`**. Luego se elige **`Editor de Visual Basic`**. Nuevamente, en el menú de la ventana que se abre, se elige **`Insertar`**, para luego seleccionar **`Módulo`** y escribir el siguiente código:

Código VBA E.2:

```
Function P(t)
  P0 = Cells(2, 3)       rem P0 está en la celda C2
  LimPobl = Cells(4, 3) rem K  está en la celda C4
  k = Cells(5, 3)        rem k  está en la celda C5
  A = (LimPobl - P0) / P0
  P = LimPobl / (1 + A * Exp(-k * t))
End Function
```

De esta forma, la ventana de edición de Visual Basic, quedaría así:



3. Una vez que ha sido editado el código de la macro, se guarda y salimos del ambiente de programación en Visual Basic para volver a la hoja electrónica de donde partimos. Este retorno se hace siguiendo el menú **`Archivo`** y seleccionando **`Cerrar y Volver a Excel`**.

4. Para evaluar la función $P(t)$ en los valores de t que están en la fila que inicia en F2, nos situamos en la celda F3 y se digita `+P(F2)`. Luego se arrastra hasta J2 para copiar la fórmula, quedando:

F3 = =+P(F2)

	A	B	C	D	E	F	G	H	I	J
1										
2	Población inicial	P0 =	100		t	0	5	10	15	20
3	Incremento	h =	5		P(t)	100	142.18925	198.2568985	269.487452	354.9789231
4		K =	1000							
5		k =	0.08							

E.4.4 Algunas Propiedades de las Celdas

Tamaño de fuente: `Cells(5, 3).Font.Size = 14`

Itálica: `Cells(5, 3)).Font.Italic = True`

Celda `Cells(n + 1, 1).Address` (devuelve por ejemplo "**`$A5`**")

Color: `Cells(1, 1).Interior.ColorIndex = 3`

E.5 Elementos de programación en VBA

Un programa computacional escrito mediante cualquier lenguaje de programación puede verse a grandes rasgos como un flujo de datos, algunos jugando el papel de datos de entrada, otros son datos que cumplen una función temporal dentro del programa y otros son datos de salida. A lo largo del programa es muy frecuente la entrada en acción de otros programas o procesos. A mayor complejidad del problema que resuelve el programa, mayor es la necesidad de programar por aparte algunos segmentos de instrucciones que se especializan en una tarea o conjunto de tareas.

Hay tres tipos de estructuras básicas que son muy utilizadas en la programación de un algoritmo, a saber, la estructura secuencial, la estructura condicional y la repetitiva.

A continuación se explica, con ejemplos programados como macros de Excel, estas estructuras. También se incluyen los programas en seudocódigo y diagramas de flujo para explicar de un modo más gráfico la lógica del programa. El uso de estos últimos es cada vez menor, pues el seudocódigo por lo general es suficientemente claro y se escribe en lenguaje muy cercano al lenguaje natural.

E.5.1 Flujo secuencial

El flujo secuencial consiste en seguir una secuencia de pasos que siguen un orden predeterminado.

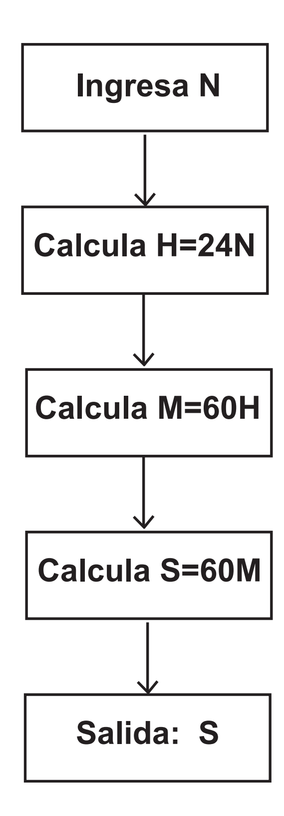
Un programa que tome un número N de días y calcule la cantidad de segundos que hay en esta cantidad de días. Este programa se puede ver como una secuencia de varios pasos:

- Inicio: Ingresa el número N de días
- Paso 1: H = 24*N, para determinar la cantidad de horas
- Paso 2: M = 60*H, para determinar la cantidad de minutos.
- Paso 3: S = 60*M, para determinar la cantidad de segundos.
- Paso 4: Retorne S.
- Fin.

La macro correspondiente a esta secuencia de cálculos puede escribirse como sigue:

Código VBA E.3:

```
Function CalculeSegundos(Dias)
    CantHoras = 24 * Dias
    CantMinutos = 60 * CantHoras
    CalculeSegundos = 60 * CantMinutos
End Function
```



E.5.2 Flujo condicional (`If–Then–Else`)

Un flujo condicional se presenta en un programa o procedimiento que debe escoger una acción o proceso a ejecutar, dependiendo de condiciones que puedan cumplirse.

El caso más sencillo ocurre cuando el programa verifica si una condición se cumple y en caso de ser verdadera ejecuta un proceso, en tanto que si es falsa ejecuta otro proceso.

En VBA tenemos la instrucción

`If...Then...Else`

Ejecuta condicionalmente un grupo de instrucciones, dependiendo del valor de una expresión.

Sintaxis

`If` *condición* **`Then`**
instrucciones

`Else`
instrucciones-else

`End If`

En algunos casos es útil seguir una sintaxis en formato de bloque:

```
If condición Then
instrucciones

ElseIf condición Then
instrucciones-elseif

...
Else
instrucciones-else
End If
```

Nota: En la ayuda del editor de Visual Basic, tenemos acceso a la referencia del lenguaje.

Usando `If`. En este ejemplo veremos cómo usar la instrucción `If...Then...Else`

Implementar un programa que calcule aproximaciones de $\sqrt{a}$. Para esto usaremos la sucesión $\{x_n\}_{n\in\mathbb{N}}$ definida en forma recurrente mediante la relación[2]:

$$\begin{cases} x_{n+1} = \frac{1}{2}(x_n + \frac{a}{x_n}) \\ x_0 = 1 \end{cases}$$

Esta sucesión converge a $\sqrt{a}$, $a \geq 0$.

El programa deberá estimar el error absoluto de las aproximaciones y será capaz de escribir un mensaje de éxito o de fracaso, dependiendo de si el error absoluto es o no menor que una tolerancia dada.

[2]Como veremos en otro capítulo, esta sucesión se obtiene buscando una solución aproximada de la ecuación $x^2 - a = 0$ aplicando el Método de Newton a la función $f(x) = x^2 - a$, con una aproximación inicial, digamos, $x_0 = 1$. La convergencia de esta sucesión es cuadrática; es decir, que a partir de cierto momento el valor x_{n+1} calculado duplica en decimales exactos al valor x_n que le precede.

	A	B	C	D
1			Tolerancia:	0,000000001
2			**Error**	
3			**absoluto**	**Se aproximó**
4	**n**	x_n	E_n	**a la toleranci**
5	0	1,000000000000000	0,414213562	FRACASO
6	1	1,500000000000000	0,085786438	FRACASO
7	2	1,416666666666670	0,002453104	FRACASO
8	3	1,414215686274510	2,1239E-06	FRACASO
9	4	1,414213562374690	1,59472E-12	FRACASO
10	5	1,414213562373090	2,22045E-16	EXITO

La figura anterior corresponde a las aproximaciones de $\sqrt{2}$ y se han obtenido mediante la evaluación de las siguientes funciones:

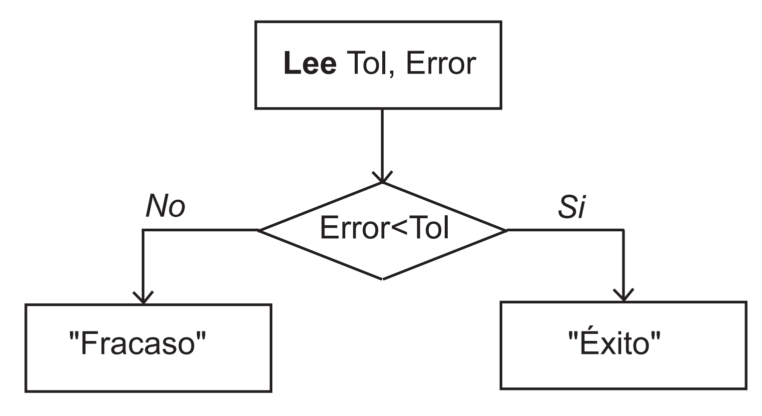
Código VBA E.4:

```
Function AproxDeRaiz(x,a)
   AproxDeRaiz = (1 / 2) * (x + a / x)
End Function

Function CalculoElError(Aproximacion, ValorExacto)
   CalculoElError = Abs(Aproximacion - ValorExacto)
End Function

Function verificaTol(Error, Tol)
   If (Error < Tol) Then
     verificaTol = "EXITO"
   Else
     verificaTol = "FRACASO"
   End If
End Function
```

El diagrama siguiente ilustra la forma en que esta última función de verificación actúa con base en el valor de sus dos parámetros de entrada:



E.5.3 Flujo repetitivo (`For-Next, While-Wend, Do While-Loop`)

El flujo repetitivo se presenta cuando se requiere la ejecución de un proceso o parte de un proceso sucesivamente, hasta que ocurra una condición que permita terminar.

Los flujos repetitivos se presentan en tres formas que obedecen a maneras diferentes de razonarlos pero que en el fondo hacen lo mismo:

- Utilizar un contador que empiece en un número y termine en otro, ejecutando el proceso cada vez que el contador tome un valor distinto.
- Mientras una condición sea verdadera, ejecutar un proceso y regresar a la condición.
- Ejecutar un proceso, hasta que una condición deje de cumplirse.

En VBA tenemos las siguientes instrucciones para realizar procesos iterativos:

1. **`For ... Next`**

   Repite un grupo de instrucciones un número especificado de veces.

   Sintaxis

   `For` *contador* = *inicio* **`To`** *fin* [**`Step`** *incremento*]
   instrucciones
   `[Exit For]`
   instrucciones
   `Next` *contador*

   (Las instrucciones entre corchetes son optativas.)

2. **`While...Wend`**

   Ejecuta una serie de instrucciones mientras una condición dada sea verdadera.

   Sintaxis

   `While` *condición*
   instrucciones
   `Wend`

   Nota: No hay un **`Exit While`**. En una subrutina, si fuera necesario, se podría usar **`Exit Sub`**

3. Una instrucción muy parecida a **`While`** pero más eficiente es **`Do`**

   Sintaxis

```
Do while condición
instrucciones
 [Exit Do]
 Loop
```

Para ilustrar estas formas de realizar un flujo repetitivo, vamos a aproximar la suma de una serie alternada de tal forma que el error no sobrepase un valor de tolerancia dado.

Consideremos la serie alternada

$$\sum_{k=1}^{\infty}(-1)^k\frac{1}{k^2} = -1+\frac{1}{4}-\frac{1}{9}+\frac{1}{16}-\ldots$$

La suma parcial N-ésima viene dada por

$$S_N = \sum_{k=1}^{N}(-1)^k\frac{1}{k^2} = -1+\frac{1}{4}-\frac{1}{9}+\frac{1}{16}-\cdots+(-1)^N\frac{1}{N^2}$$

es decir

$$\begin{array}{lcll} S_1 & = & -1 & = -1 \\ S_2 & = & -1+1/4 & = -0.75 \\ S_3 & = & -1+1/4-1/9 & = -0.861111... \\ S_3 & = & -1+1/4-1/9+1/16 & = -0.798611... \\ & & \vdots & \end{array}$$

De acuerdo con la teoría de series alternadas, la serie $\sum_{k=1}^{\infty}(-1)^k\frac{1}{k^2}$ converge a un valor S. Al aproximarla con la suma parcial S_N, el error de aproximación es menor que $\frac{1}{(N+1)^2}$, es decir

$$|S-S_N| < \frac{1}{(N+1)^2}$$

Primer problema

Dada una tolerancia **TOL**, (donde TOL es una cantidad positiva, como 10^{-1}, 5×10^{-8}, etc.), calcular cada una de las sumas parciales hasta que el error de aproximación sea menor que TOL.

La cantidad $\frac{1}{(N+1)^2}$ juega en este problema el papel de una cota del error, al aproximar la serie correspondiente hasta el término N-ésimo.

Solución

	A	B	C	D	E	F	G	H
31			Suma de la Serie					
32		TOL	N	Suma-Parcial N	Error estimado	ya?		
33		0,01	1	-1	0,25	Error estimado > .01		
34			2	-0,75	0,111111111	Error estimado > .01		
35			3	-0,861111111	0,0625	Error estimado > .01		
36			4	-0,798611111	0,04	Error estimado > .01		
37			5	-0,838611111	0,027777778	Error estimado > .01		
38			6	-0,810833333	0,020408163	Error estimado > .01		
39			7	-0,831241497	0,015625	Error estimado > .01		
40			8	-0,815616497	0,012345679	Error estimado > .01		
41			100	-0,822417533	9,80296E-05	OK, error estimado < .01		

Podemos implementar dos macros: una para el cálculo de las sumas parciales y otra para hacer la verificación del error estimado. En este caso, vamos a suponer que **TOL** está en la celda **B33**

Código VBA E.5:

```
Function sumaParcial(hastaN)
  Dim Acum,
  Dim signo As Integer
  Acum = 0 signo = -1

  For k = 1 To hastaN
    Acum = Acum + signo * 1 / k ^ 2
    signo = -signo
  Next k
  sumaParcial = Acum
  End Function
rem -----------------------------------------------------------------
Function verificaTol(N, Tol)
   If (1 / (N + 1) ^ 2 > Tol) Then
     verificaTol = "Error estimado > " + Str(Tol)     rem Tol es un número
   Else                                                rem no una String;
     verificaTol = "OK, error estimado <" + Str(Tol) rem por esto debemos usar rem Strrem
   End If
End Function
```

En la primera llamada de las macros se usó `sumaParcial(C33)` y `verificaTol(C33;B$33)`

Segundo problema

Dada una tolerancia *TOL*, aproximar la suma de la serie con una cantidad N de términos lo suficientemente grande de tal manera que $\dfrac{1}{(N+1)^2} < TOL$.

1. **Primera solución**:

   Dado que hay que sumar hasta el término N-ésimo tal que $\frac{1}{(N+1)^2} < TOL$, en este caso es posible despejar el entero positivo N, quedando:

$$N > \sqrt{\frac{1}{TOL}} - 1$$

   Tomamos N como la parte entera superior de $\sqrt{\frac{1}{TOL}} - 1$, lo que equivale a calcular la parte entera de $\sqrt{\frac{1}{TOL}}$; es decir,

$$N = \left\lfloor \sqrt{\frac{1}{TOL}} \right\rfloor \quad \text{(parte entera)}$$

   Los pasos a seguir para programar la suma a partir de la tolerancia dada, son los siguientes:

   Inicio: Ingresa la tolerancia con que se hará la aproximación.

   Paso 1: Calcular $N = \left\lfloor \sqrt{\frac{1}{TOL}} \right\rfloor$

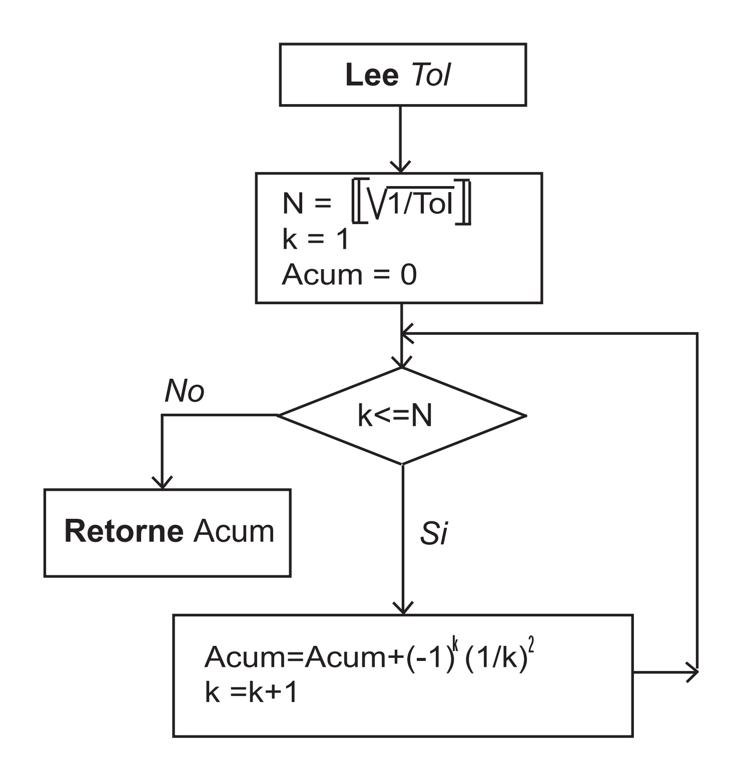
   Paso 2: `Acum = 0` (Se iniciliza el acumulador para la suma).

   Paso 3: Para `k = 1`...`N`:

   `Acum = Acum +` $(-1)^k \frac{1}{k^2}$

   Paso 4: Retorne Acum.

   Fin.



Observe que en cada sumando, se incluye el factor $(-1)^k$, que en la práctica, más que una potencia, lo que indica es el cambio de signo en los términos sucesivos. Para evitarle al programa cálculos innecesarios, podemos inicializar una variable *signo* en -1 y en cada paso de la instrucción repetitiva se cambia al signo contrario.

	A	B	C	D	E
1					
2		Aproximación de una serie alternada			
3	Primera solución: USANDO FOR K=1 TO N				
4		Tol	N	Suma Parcial	Error estimado
5		1	1	-1,00000000000000	0,25
6		0,1	3	-0,86111111111111	0,0625
7		0,01	10	-0,82796217561099	0,008264463
8		0,001	31	-0,82297055860543	0,000976563
9		0,0001	100	-0,82251753337413	9,80296E-05
10		0,00001	316	-0,82246204205917	9,95134E-06
11		0,000001	1000	-0,82246753392411	9,98003E-07
12		0,0000001	3162	-0,82246698343114	9,99543E-08
13		0,00000001	10000	-0,82246703842461	9,998E-09
14		1E-09	31622	-0,82246703292409	9,99986E-10
15		1E-10	100000	-0,82246703347410	9,9998E-11

La macro correspondiente a este programa puede escribirse como sigue:

Código VBA E.6:

```
Function SumaParcial(Tol)
    Acum = 0 signo = -1
    N = Int(1 / Sqr(Tol))
    For k = 1 To N
        Acum = Acum + signo * 1 / k ^ 2
        signo = -signo
    Next k
    SumaParcial = Acum
End Function
```

2. **Segunda solución:**

En esta solución no es necesario calcular el valor de N, sino que se suman los términos mientras no se haya alcanzado la tolerancia. El programa en seudocódigo se puede escribir como sigue:

Inicio: Ingresa la tolerancia con que se hará la aproximación.

Paso 1: Iniciar con $N = 1$.

Paso 2: **`Acum = -1`** (Se iniciliza el acumulador para la suma con el primer término).

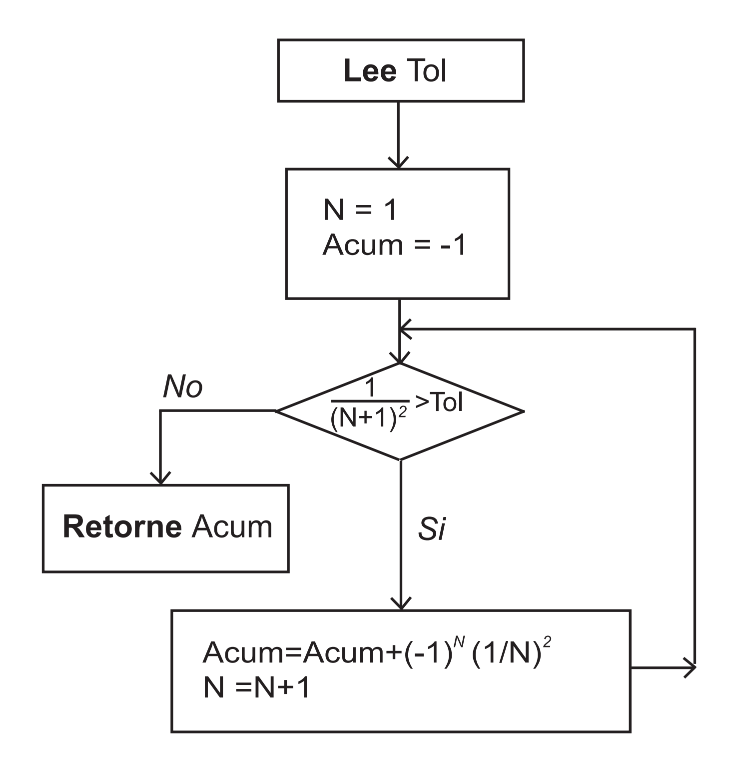
Paso 3: Mientras $\frac{1}{(N+1)^2} > Tol$:

. $N = N+1$

. **Acum = Acum +** $(-1)^N \frac{1}{N^2}$

Paso 4: Retorne **Acum**.

Fin .



El código es

Código VBA E.7:

```
Function SumaParcial2(Tol)
    N = 1 Acum = -1 signo = 1
    While (1 / (N + 1) ^ 2 > Tol)
          N = N + 1
          Acum = Acum + signo * 1 / N ^ 2
          signo = -signo
    Wend
    SumaParcial2 = Acum
End Function
```

E.5.4 Declaración de variables en un programa

Las variables en un programa pueden ser declaradas de un tipo explícito, dependiendo de si es de tipo numérico, tira de caracteres (*String*), etc. Cada tipo de variable ocupa una cantidad de memoria diferente, por lo que es importante

en un programa hacer una definición apropiada de los tipos de variable, con el fin de que la memoria sea utilizada eficientemente y los programas sean corridos a una velocidad adecuada.
En caso de que a una variable no se declare un tipo de datos,el compilador le asigna un tipo especial llamado **Variant**. Este tipo admite datos numéricos, de cadena, de fecha, así como tipos definidos por el usuario y los valores especiales **Empty** y **Null**.
Las variables pueden declararse en tres lugares diferentes.

- A las variables que son declaradas dentro de una función o subrutina se les llama *variables locales*. Estas sólo pueden ser utilizadas en instrucciones que estén dentro de esa función o subrutina. Una variable local solo es conocida y utilizable dentro del código de esa función o subrutina y es desconocida por cualquier otra función o subrutina.

  Por ejemplo, en la siguiente función, las variables **`Suma`** y **`a`** son variables locales:

Código VBA E.8:

```
Function MiSuma(x)
   Dim a as Double
   a=1
   Suma = a + x
   MiSuma = Suma
End Function
```

- También se puede declarar una variable como parametro de una función o subrutina. Al ser invocada esta función, este parámetro debe tener un valor específico y luego puede actuar como variable local en esa función o subrutina. En el ejemplo anterior, la variable **`x`** sirve como parámetro de la función **`MiSuma`**.

- También se puede declarar una variable fuera de todas las funciones o subrutinas. A este tipo de variables se les llama *variables globales* y pueden ser utilizadas y alteradas por cualquier función o subrutina del programa.

  En el siguiente ejemplo, la Variable *Acumulador* es global

Código VBA E.9:

```
Dim Acumulador as Double

Function Suma2(t)
   Acumulador=2
   Suma2 = 2*Acumulador + t
End Function

Sub OtrasOperaciones()
   M=Suma2(5)
   Acumulador=3*Acumulador
End Sub
```

Al correr esta última subrutina, la variable local **`M`** toma el valor de 9 a nivel local y desaparece una vez que se ha ejecutado la segunda instrucción de la subrutina, mientras que la variable **`Acumulador`** toma por primera vez el valor de 2, al ser definido por la función **`Suma2`**. Luego este valor de 2 es multiplicado por 3 en la subrutina **`OtrasOPeraciones`**, finalizando con un valor de 6, que no desaparece cuando han terminado las instrucciones de dicha subrutina.

El siguiente es otro ejemplo de uso de una variable global, que incluye un proceso previo de borrado.

Limpiando celdas entre diferentes corridas de un programa.

	A	B	C
1	N	= 25	Divisores
2			1
3			5
4			25
5			

Dado un número entero positivo N que se coloca en la celda B1, el programa calcula todos los divisores de N, incluyendo el 1 y el mismo número N, y los imprime en la tercera columna. Se ha definido una variable global llamada **Conta** que servirá para contar la cantidad de divisores que tiene N. Al cambiar el valor de N y ejecutar de nuevo el programa, la variable global tiene almacenada el número de celdas utilizadas en la corrida previa. En la primera ejecución del programa, la variable global tiene almacenado un valor de cero, por lo que no ejecuta ningún borrado.

El código de este programa es:

Código VBA E.10:

```
Dim Conta As Long

Sub ImprimirDivisores()
    dim k as Long
    N = Cells(1, 2)
    rem borrar celdas hasta donde imprimió la vez anterior
     For k = 1 To Conta
       Cells(1 + k, 3) = Null
     Next k

   Conta = 0
   For k = 1 To N
     If ((N Mod k) = 0) Then
        Conta = Conta + 1
        Cells(1 + Conta, 3) = k
     End If
   Next k
End Sub
```

E.5.5 Tipos de datos

La siguiente tabla describe algunos de los principales tipos de datos y que son de interés en el presente material:

Tipo	Memoria	Rango Aproximado
Byte	1 byte	0...255
Boolean	2 bytes	True, False
Integer	2 bytes	-32763...32767
Long (entero grande)	4 bytes	-2147483648...2147483647
Single (punto flotante de precisión simple)	4 bytes	$-3.4\times10^{38}\cdots-1.4\times10^{-45}$ para valores negativos $1.4\times10^{-45}\cdots3.4\times10^{38}$ para valores positivos
Double (punto flotante de doble precisión)	8 bytes	$4.9\times10^{-324}\cdots-4.9\times10^{-324}$ para valores negativos $1.8\times10^{308}\cdots3.4\times10^{38}$ para valores positivos
Variant (numérico)	16 bytes	Cualquier valor numérico hasta el tamaño del tipo Double

E.5.6 Manejo de rangos

Un rango en Excel corresponde a una selección de celdas. Una *selección* de las celdas de una fila o una columna se maneja en Excel como una matriz de orden $1\times n$ o de orden $n\times 1$ (un vector). La *selección* de un bloque de celdas se maneja como una matriz $n\times m$. Si una celda está en blanco, se lee un cero.

Calculando un promedio simple. Consideremos una tabla con 5 notas, todas con igual peso.

A	B	C	D	E	F	G	H
	Promedio Simple						
	Nombre	N1	N2	N3	N4	N5	Promedio
	Pedro Pérez	40	85	90	80	30	65
	Ana Pereira	55	66	77	89	100	77,4
	Victoria Ching	100	100	0	100	90	78

Para calcular el promedio simple, en cada fila, vamos a hacer una macro que recibe un rango, cuenta las notas, suma y divide entre el número de notas.

Código VBA E.11:

```
Function PromedioSimple(R As Range) As Double
    rem R es la variable que recibe el rango
    Dim n As Integer
    Dim suma As Double

    suma = 0
    n = R.EntireColumn.Count           rem cantidad de notas en el rango
    For Each x In R                     rem suma de las notas
        suma = suma + x
    Next x
```

```
    PromedioSimple = suma / n                    rem promedio simple
End Function
```

En la primera celda de la columna **Promedio**, llamamos a la macro con: `PROMEDIO(C63:G63)` pues en este caso el rango es **C63:G63**.

Calculando el promedio de varias calificaciones eliminando las dos más bajas. En este caso, a un conjunto de calificaciones les calculamos el promedio simple pero eliminando las dos notas más bajas. El programa **PromedioQ** suma las n notas de una fila (rango), localiza la posición (en el vector **R**) de las dos notas más bajas y luego le resta a la suma estas dos notas para luego dividir entre $n-2$. En este caso, el rango **R** es una matriz $1 \times n$, o sea, se puede ver como un vector de n componentes.

H65 =PromedioQ(C65:G65)

	A	B	C	D	E	F	G	H
60		Promedio de notas excluyendo las dos más bajas						
61								
62		Nombre	N1	N2	N3	N4	N5	Promedio
63		Pedro Pérez	40	85	90	80	30	85
64		Ana Pereira	55	66	77	89	100	88,66667
65		Victoria Ching	100	100	0	100	90	100

Código VBA E.12:

```
Function PromedioQ(R As Range) As Double

    Dim n, i, Imin1, Imin2 As Integer
    Dim suma As Double

    suma = 0
    n = R.EntireColumn.Count rem número de elementos de la selección

    For i = 1 To n
      suma = suma + R(1, i)   rem R es una matriz 1xn (o sea, un vector)
    Next i                     rem En R no se hace referencia a la celda

    Imin1 = 1                 rem POSICION de la primera nota mínima en R
    For i = 1 To n
      If R(1, i) < R(1, Imin1) Then
        Imin1 = i
      End If
    Next i

    Imin2 = 1                 rem POSICION de la segunda nota mínima en R
    If Imin1 = 1 Then
```

```
      Imin2 = 2
   End If

   rem comparar con todos excepto Imin
   For i = 1 To n
   If (R(1, i) < R(1, Imin2)) And (i <> Imin1) Then
       Imin2 = i
   End If
   Next i

   PromedioQ = (suma - R(1, Imin1) - R(1, Imin2)) / (n - 2)
End Function
```

Nota: También podríamos resolver este problema usando **Selection.Sort** pero la programación es un poco más compleja.

E.5.7 Subrutinas. Edición y ejecución de una subrutina

Las subrutinas o procedimientos es otro de los tipos básicos de programas en Visual Basic. Una descripción de la sintaxis de una subrutina que no es completa, pero sí suficiente para los alcances de este libro es la siguiente:

Sintaxis:

Sub Nombre_de_Subrutina ([*lista-argumentos*])

[*instrucciones*]

End Sub

o también

[Private | Public] [Static] Sub Nombre-de-Subrutina([*lista-argumentos*])

[*instrucciones*]

End Sub

Además:

Public. Es opcional. Indica que la subrutina puede ser llamada por todas las demás subrutinas sin importar donde se encuentre.

Private. Es opcional. Indica que la subrutina puede ser llamada solamente por otras subrutinas que se encuentren en el mismo módulo[3].

Static. Es opcional. Indica que las variables locales de la subrutina se mantienen constantes de una llamada a otra. El ámbito de acción de esta declaración no incluye a variables declaradas fuera de la subrutina.

Nombre_De_Subrutina. Es requerido. Indica el nombre de la subrutina.

lista-argumentos. Es opcional e indica las variables que conforman los argumentos con que una subrutina es llamada. Para separar una variable de otra se escribe una coma.

instrucciones. Es opcional y conforma el conjunto de instrucciones que son ejecutadas a lo largo de la subrutina.

Elevar al cuadrado los valores de una selección y ejecutar desde la ventana de ejecución de macros.

	A	B	C
1			
2	-1	2	9
3	-2	3	10
4	-3	4	11
5	-4	5	12
6	-5	6	
7	-6	7	
8	-7	8	
9	-9		
10	-10		

→

	A	B	C
1			
2	1	4	81
3	4	9	100
4	9	16	121
5	16	25	144
6	25	36	0
7	36	49	0
8	49	64	0
9	81	0	0
10	100	0	0

Podemos implementar una subrutina en una hoja, que recorra una *selección* hecha con el *mouse* y que eleve al cuadrado el valor de cada celda.

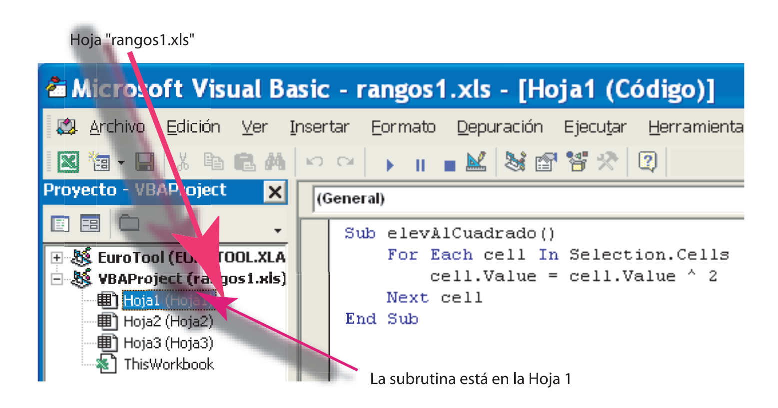
Código VBA E.13:

```
Sub elevAlCuadado()
    For Each cell In Selection.Cells  rem para cada celda de la selección
        cell.Value = cell.Value ^ 2   rem recalcula el valor de la celda
    Next cell
End Sub
```

Nota: La macro se aplica a los datos que están actualmente seleccionados.

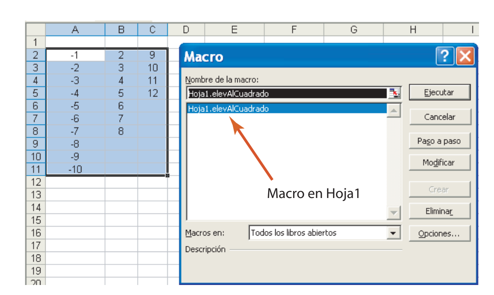
[3]Aparte de la posibilida de declarar la subrutina como *Private* o *Public*, también declarar como *Friend*, pero esto tiene que ver más con la programación orientada a objetos

• Para editar la subrutina, vamos al editor VB (**`Alt-F11`**) y hacemos doble-clic sobre **`(Hoja1)`**



Escribimos el código, compilamos (en menú **`Depuración`**), guardamos y nos devolvemos a la hoja.

• Para ejecutar la macro seleccionamos la tabla con el *mouse* y levantamos la ventana de ejecución de macros (**`Alt-F8`**) y damos *clic* en "EJECUTAR"



Nota: Esta subrutina también se puede editar en un módulo. Para ejecutarla se procede de la misma forma.

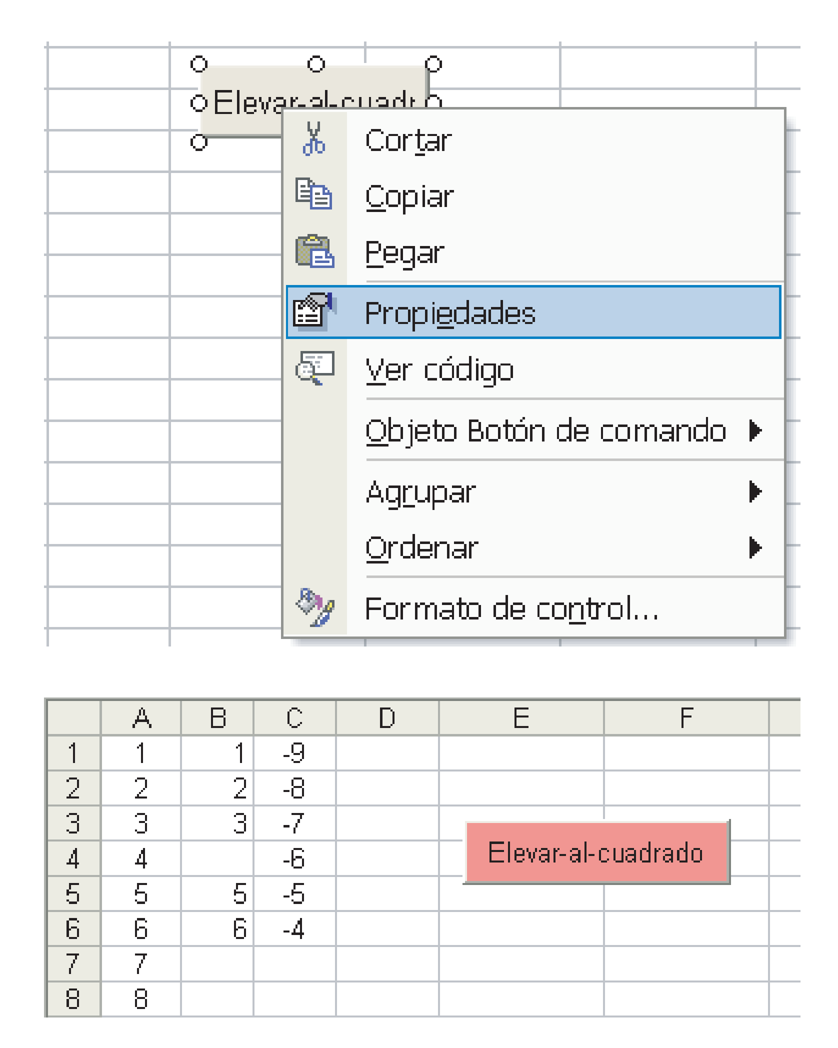
E.5.8 Ejecución de una subrutina mediante un botón

Otra posibilidad bastante práctica para ejecutar un programa o subrutina como, los presentados en la sección precedente, es mediante un botón de comando.

Elevar al cuadrado los valores de una selección. Ejecutar mediante un botón.

1. Primero digitamos la tabla de valores. Luego insertamos un botón. Para ésto seleccionamos un botón de comando del cuadro de controles (si la barra no está disponible, puede habilitarla con **`Ver - Barra de herramientas - Cuadro de Controles`**).

2. Luego *hacemos clic en el lugar de la hoja donde queremos el botón*. Una vez que tenemos el botón, podemos agregar algunas propiedades como etiqueta, color de fondo, etc., en el menú de contexto. Este menú se abre con **`clic derecho + propiedades`**. Luego cerramos el menú.



3. Para editar el código que deberá ejecutar el botón, le damos un doble *clic* al botón (que todavía está en modo diseño). En este caso, si es la primera vez, nos aparece el código:

Código VBA E.14:

```
Private Sub CommandButton1_Click() End Sub
```

Aquí tenemos dos opciones:

a.) Implementar la subrutina por separado y luego llamarla desde la subrutina del botón.

Código VBA E.15:

```
 Sub elevAlCuadado()
   For Each cell In Selection.Cells  rem para cada celda de la selección
       cell.Value = cell.Value ^ 2   rem recalcula el valor de la celda
   Next cell
End Sub

rem -----------------------------------------------------------------------
Private Sub CommandButton1_Click()
   elevAlCuadado    rem sin paréntesis
End Sub
rem -----------------------------------------------------------------------
```

b.) Incluir en la subrutina del botón lo que queremos que se ejecute cuando se haga clic sobre él.

Código VBA E.16:

```
Private Sub CommandButton1_Click()
    For Each cell In Selection.Cells  rem para cada celda de la selección
       cell.Value = cell.Value ^ 2    rem recalcula el valor de la celda
   Next cell
End Sub
```

4. Una vez que escogemos alguna de las dos opciones anteriores, compilamos (en menú **`Depurar`**), guardamos y nos devolvemos a la hoja.

5. Para habilitar el botón debemos deshabiltar el ícono de diseño .

6. Ahora solo resta *seleccionar* la tabla y hacer clic sobre el botón.

Observe que al dar un clic sobre el botón, el programa opera sobre lo que *tengamos seleccionado* previamente

	A	B	C	D	E	F
1	1	1	-9			
2	2	2	-8			
3	3	3	-7			
4	4		-6			
5	5	5	-5			
6	6	6	-4			
7	7					
8	8					

Elevar-al-cuadrado

E.5.9 Matrices dinámicas

Cuando hacemos una *selección* con el mouse, es conveniente ingresar los valores seleccionados en una matriz dinámica, es decir, una matriz que se ajuste a la cantidad de datos seleccionada y que, eventualmente, se pueda recortar o hacer más grande.

Una matriz dinámica **`mtr1`** de entradas enteras se declara así:

```
Dim mtr1() As Integer rem Declara una matriz dinámica.
```

Las instrucciones siguientes le dan un tamaño de la matriz **`mtr1`** y la inicializa. Observe el uso de **`Redim`** para cambiar el tamaño de la matriz dinámica.

Código VBA E.17:

```
Dim   mtr1() As Integer    rem Declara una matriz dinámica mtr1. Dim
r() as Double              rem Declara una matriz dinámica r.
%Preguntar que hacer aquí
Redim  mtr1(10)            rem Cambia el tamaño a 10 elementos,
1x10. Redim r(n,m)         rem Cambia tamaño a   n x m

For i = 1 To 10            rem Bucle 10 veces.
    mtr1(i) = i           rem Inicializa la matriz.
Next i
```

Usando **`Preserve`** se puede cambiar el tamaño de la matriz **`mtr1`** pero sin borrar los elementos anteriores.

```
Redim Preserve mtr1(15) rem Cambia el tamaño a 15 elementos.
```

Centro de gravedad de un conjunto de puntos en el plano cartesiano. Consideremos un conjunto $\Omega = \{(x_i,y_i)\ /\ i=1,2,\cdots,n,\ x_i,y_i \in \mathbb{R}\}$. Supongamos que a cada punto (x_i,y_i) se le asigna un peso p_i tal que $\sum_{i=1}^{n} p_i = 1$.

El centro de gravedad G_Ω de Ω se define así:

$$G_\Omega = \sum_{i=1}^{n} p_i(x_i,y_i) = \left(\sum_{i=1}^{n} p_i x_i, \sum_{i=1}^{n} p_i y_i\right)$$

Por ejemplo, si tenemos dos puntos $A(x_1,y_1), B(x_2,y_2) \in \mathbb{R}^2$ con igual peso (este deberá ser $1/2$ para cada punto), entonces el centro de gravedad es el punto medio del segmento que une A con B

$$G_\Omega = \frac{1}{2}(x_1,y1) + \frac{1}{2}(x_2,y_2) = \left(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}\right)$$

La subrutina que calcula el centro de gravedad de un conjunto de puntos $\Omega \subset \mathbb{R}^2$ actúa sobre un rango de tres columnas en el que se han escrito las coordenadas (x,y) de dichos puntos, así como sus pesos. Podemos correr el programa una vez que se ha *seleccionado* el rango completo en el que se ubican los puntos y sus pesos, haciendo clic sobre un botón “Centro Gravedad". El gráfico es un trabajo adicional que se puede hacer con el asistente de gráficos.

Como el programa necesita que el usuario haya seleccionado al menos dos filas y exactamente tres columnas, incluimos un fragmento de código adicional para controlar la selección.

Código VBA E.18:

```
Set R = Selection n = R.Rows.Count     rem  Número de filas m =
R.Columns.Count  rem  Número de columnas

If n > 1 And m = 3 Then
      rem todo está bien, el programa continúa
Else
     MsgBox ("Debe seleccionar los datos")
     Exit Sub          rem  salimos de la subrutina
End If
```

Observemos que si no se cumple el requisito, se envía un mensaje y la subrutina se deja de ejecutar. Puede causar curiosidad que antes del **Else** no haya código. A veces se usa esta construcción por comodidad. Si no hay código el programa continúa.

Veamos el código completo en la subrutina del botón

Código VBA E.19:

```
Private Sub CommandButton2_Click()
    Dim R As Range
    Dim n,m, i As Integer
    Dim x() As Double   rem  matriz dinámica
    Dim y() As Double   rem  se ajustará a la selección de datos
    Dim p() As Double
    Dim Sumapesos, Gx, Gy As Double
```

```
        rem En el rango R guardamos el rango seleccionado:
    Set R = Selection
    n = R.Rows.Count     rem  Número de filas
    m = R.Columns.Count  rem  Número de columnas
                         rem  chequear que se hayan seleccionado los datos de la tabla
    If n > 1 And m = 3 Then
       rem todo está bien, el programa continua
    Else
      MsgBox ("Debe seleccionar los datos")
      rem  salimos de la subrutina
      Exit Sub
    End If

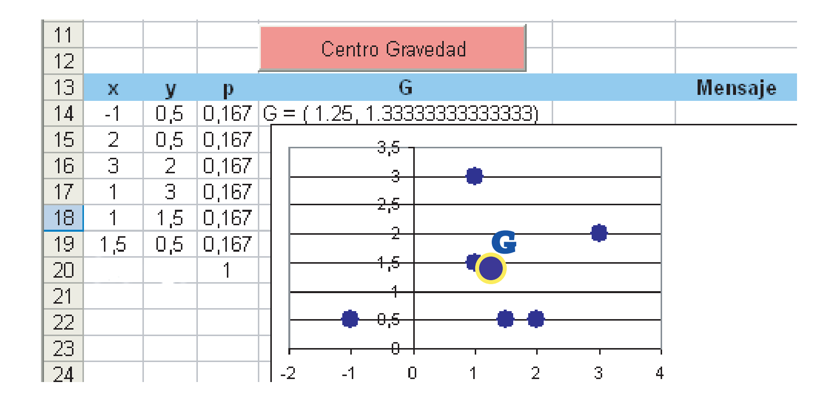
    ReDim x(n)         rem  vector x tiene ahora n campos
    ReDim y(n)
    ReDim p(n)

    Sumapesos = 0      rem inicializa para acumular
    Gx = 0
    Gy = 0
                       rem inicializa las matrices con los datos

        For i = 1 To n
            x(i) = R(i, 1) rem entra los datos de columna de los xrem s
            y(i) = R(i, 2)
            p(i) = R(i, 3)
                           rem calcula centro de gravedad
            Sumapesos = Sumapesos + p(i)
            Gx = Gx + x(i) * p(i)
            Gy = Gy + y(i) * p(i)
        Next i
        If Abs(Sumapesos - 1) < 0.001 Then  rem la suma de los pesos debe ser 1
            Gx = Gx / Sumapesos
            Gy = Gy / Sumapesos
                               rem escribe G en la celda D14
            Cells(14, 4) = "G = (" + Str(Gx) + "," + Str(Gy) + ")"
            Cells(14, 5) = ""  rem limpia E14 de mensajes previos
        Else
                               rem mensaje de error
            Cells(14, 5) = "Error, los pesos  suman " + Str(Sumapesos)
            Cells(14, 4) = ""  rem limpia D14 de valores previos
        End If
End Sub
```

Nota: Por cuestiones de redondeo, la instrucción **`If Sumapesos = 1`** se cambió por **`If Abs(Sumapesos - 1) < 0.001`** .

Un ejemplo de corrida se ve en la figura que sigue



	x	y	p	G	Mensaje
14	-1	0,5	0,167	G = (1.25, 1.33333333333333)	
15	2	0,5	0,167		
16	3	2	0,167		
17	1	3	0,167		
18	1	1,5	0,167		
19	1,5	0,5	0,167		
20			1		

EJERCICIOS

1. Usando la notación del último ejemplo, se define la *inercia total* de Ω como

$$I(\Omega) = \sum_{i=1}^{n} p_i \,||\,(x_i, y_i) - G_\Omega\,||^2$$

donde $||(a,b)|| = \sqrt{a^2+b^2}$ es la norma usual.

Implemente una subrutina que calcula la Inercia Total de Ω y aplíquela a la tabla del ejemplo 8.

2. Si tenemos puntos $\{(x_1, y_1), (x_2, y_2), \cdots (x_n, y_n)\}$, se define

$$L_{in}(x) = \frac{(x-x_1)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_1)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)}$$

O sea, en $L_{in}(x)$ se elimina el factor $(x - x_i)$ en el numerador y el factor $(x_i - x_i)$ en el denominador

Implemente una hoja, como se ve en la figura, en la que el usuario hace una selección de la tabla y al hacer *clic* en el botón, se calcula $L_{2n}(2.56)$

	A	B	C	D
1	x_i	y_i	x	$L_{2n}(x)$
2	2,2	0,3	2,56	?
3	2,3	2,3		
4	2,4	3,2		Calcular
5	2,5	2,1		

Parte del código sería

Código VBA E.20:

```
...
Set R = Selection
n = R.Rows.Count  rem  Número de filas
ReDim x(n)        rem  vector x tiene ahora n campos
ReDim y(n)        rem  vector y tiene ahora n campos
valorX = Cells(2, 3) rem  valorX está en celda C2
If n > 1 Then       rem si hay datos seleccionados, n > 1
    For i = 1 To n
        x(i) = R(i, 1) rem entra los datos de columna de los xirem s y los yirem s
        y(i) = R(i, 2)
    Next i

        L2n= 1        rem inicia cálculo de L2n(variable x)
        For j = 1 To n rem calculamos L2n(valorX)
            If j <> 2 Then
                L2n = L2n * (valorX - x(j)) / (x(k) - x(j))  rem L2n evaluado en valorX
            End If
        Next j
    Cells(2, 4) = L2n
Else
    MsgBox ("Debe seleccionar los datos")
    Exit Sub              rem  aborta la subrutina si no hay datos seleccionados
End If
...
```

3. Si tenemos n puntos $\{(x_1,y_1),(x_2,y_2),\cdots(x_n,y_n)\}$, un polinomio que pasa por todos estos puntos se llama *Polinomio Interpolante*. Un polinomio interpolante muy conocido es el *Polinomio Interpolante de Lagrange*

$$P(x) = \sum_{i=1}^{n}\left[\prod_{j=1,j\neq i}^{n}\frac{(x-x_j)}{(x_i-x_j)}\right]y_i = \sum_{i=1}^{n} L_{in}(x)\cdot y_i$$

donde $L_{in}(x) = \dfrac{(x-x_1)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_1)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)}$
o sea

$$P(x) = \sum_{i=1}^{n}\frac{(x-x_1)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)(x_i-x_1)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)}y_i$$

O sea, para cada valor de i se elimina el factor $(x-x_i)$ en el numerador y el factor (x_i-x_i) en el denominador

Este polinomio cumple $P(x_i) = y_i \quad i = 1,2,...n$.

Por ejemplo, para el caso de tres puntos (x_1,y_1), (x_2,y_2) y (x_3,y_3), el Polinomio de Lagrange es

$$P(x) = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)}y_1 + \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)}y_2 + \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_1)}y_3$$

(a) Implemente una subrutina que, a partir de una *selección* de puntos (x_i, y_i) (en dos columnas) en una hoja de Excel, *evalúa* el polinomio interpolante en una valor dado de antemano en una celda, o sea, calcula $P(a)$ para un valor a dado.

(b) Aplique la implementación anterior a una tabla como la que se presenta en la figura

Polinomio de Lagrange				
x_i	y_i	a	P(a)	
-1	0,5	3,5	?	
2	0,5			
3	2			
4	3			
5	1,5			
6	4,6			

Parte del código sería

Código VBA E.21:

```
...
suema = 0          rem inicializa para acumular
If n > 1 Then       rem si hay datos seleccionados, n > 1
    For i = 1 To n
        x(i) = R(i, 1) rem entra los datos de columna de los xirem s y los yirem s, inicia e
        y(i) = R(i, 2) rem aquí, iniciamos desde x1, es decir el x0 de la teoría, es x1
    Next i

    For i = 1 To n
        Lin = 1         rem inicia cálculo de Lkn
        For j = 1 To n rem calculamos Lkn(valorX)
            If j <> k Then
                Lin = Lin * (valorX - x(j)) / (x(i) - x(j)) rem Lin evaluado en valorX
            End If
        Next j
        suma = suma + Lin * y(i)
    Next i
    Cells(5, 4) = suma
Else
                                rem ventana de advertencia: seleccione datos!
    MsgBox ("Debe seleccionar los datos")
    Exit Sub                rem  aborta la subrutina si no hay datos seleccionados
End If
...
```

E.5.10 Inclusión de procedimientos de borrado

En ocasiones es necesario borrar alguna información que ha sido escrita en una hoja electrónica, por lo que es importante conocer una forma de incorporar en la aplicación un procedimiento de borrado.

Presentamos a continuación un programa que, a partir de un número N construye el triángulo de Pascal de N niveles. También se incluye un programa que funciona como borrador o destructor del triángulo.

En cada nivel del triángulo hay un uno en los extremos y, a partir del tercer nivel, cada número, salvo los extremos, es la suma de los dos de arriba. Concretamente, el triángulo de Pascal es un arreglo triangular de números de la forma:

```
              1
            1   1
          1   2   1
        1   3   3   1
      1   4   6   4   1
    1   5  10  10   5   1
  1   6  15  20  15   1   6   1
⋰                               ⋱
```

Por simplicidad, presentamos un programa que lee el número de niveles del triángulo de Pascal en la celda E1 y lo despliega de la forma:

Código VBA E.22:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
...
```

Tanto la construcción del triángulo como su destrucción pueden ser activados mediante botones. Las instrucciones que realizan estos procedimientos también pueden ser incluidas directamente en el código de los botones, tal y como se detalla a continuación.

El código para la construcción del triángulo quedaría así:

Código VBA E.23:

```
Private Sub EjecucionDePascal_Click()
   rem  Lectura de la cantidad de niveles:
     N = Cells(1,5)
   rem  Llenar unos:
     For i = 1 To N
         Cells(i,1)= 1
         Cells(i,i)= 1
     Next i
   rem  Llenar el resto:
```

```
    If N > 2 Then
        For i=3 To N
            For j=2 To i-1
                Cells(i,j)= Cells(i-1,j) + Cells(i-1,j-1)
            Next j
        Next i
    End If
End Sub
```

El procedimiento para borrar el triángulo también lee el número de niveles en la celda *E1* y hace el mismo recorrido de celdas que hizo el constructor y en cada celda escribe un valor nulo.

Código VBA E.24:

```
Private Sub Borrador_Click()
    N = Cells(1, 5).Value
    For i = 1 To N
        For j = 1 To i
            Cells(i, j).Value = Null
        Next j
    Next i
End Sub
```

	A	B	C	D	E	F	G	H	I	J
1	1				9					
2	1	1							Construir	
3	1	2	1							
4	1	3	3	1						
5	1	4	6	4	1				Borrar	
6	1	5	10	10	5	1				
7	1	6	15	20	15	6	1			
8	1	7	21	35	35	21	7	1		
9	1	8	28	56	70	56	28	8	1	
10										

EJERCICIOS

1. **Triángulo de Pascal.** Haga un programa que, al activarlo desde un botón de comando, lea un un número *N* de la celda A1 y construya el triángulo de Pascal en la forma:

Código VBA E.25:

```
           1
         1 1
      1  2 1
   1  3  3 1
 1 4  6  4 1
1 5 10 10 5 1 ...
```

Además, incluya un botón que active un destructor especializado para este triángulo.

2. **Aritmética entera**. En VBA hay dos tipos de división: `a/b` es la división corriente (en punto flotante) y `a\b` que es la *división entera*. Por ejemplo, `3/2 = 1.5` mientras que `3\2 = 1`.

   Hay muchos algoritmos en los que se usa exclusivamente divisón entera. Veamos un ejemplo.

   Si N es un entero positivo, vamos a calcular el entero más grande que es menor o igual a $\sqrt{N}$, es decir $\lfloor\sqrt{N}\rfloor$ (" $\lfloor\ \rfloor$ " es la parte entera). El algoritmo, que opera en aritmética entera, es el siguiente

   a.) Inicio: $s_0 = \frac{N}{2}$

   b.) $s_{i+1} = \dfrac{s_i + \dfrac{N}{s_i}}{2}, \quad i = 0,1,2,...$

   c.) iterar hasta que $s_{i+1} \geq s_i$

   d.) el último s_i es $\lfloor\sqrt{N}\rfloor$

   Por ejemplo si $N = 10$, $\sqrt{N} \approx 3{,}16227766$ y el último s_i sería 3

   Implemente el algoritmo.

3. **Números primos.** Implemente un programa para hallar el número primo más cercano a un número N, siendo N un número entero positivo mayor que uno.[4]

   Por ejemplo, los números primos menores o iguales que 50 son:

   2, 3, 5, 7, 19, 23, 29, 31, 37, 41, 43.

   Si $N = 50$, el programa debería retornar el número 43.

   Un procedimiento clásico para hallar todos los números primos menores que un entero positivo N, es la llamada *criba de Eratóstenes*. Lo que se hace es colocar en una lista todos los números del 2 al N e ir eliminando de esta lista todos los múltiplos de 2 (4, 6, 8, ...), todos los múltiplos de 3 (6, 9, 12, ...), y así sucesivamente, hasta

[4] Un número natural $p > 1$, es *primo* si sus únicos divisores positivos son 1 y p. En caso contrario, es un número *compuesto*.

eliminar todos los múltiplos de los primos que han ido quedando en la lista menores o iguales que $\sqrt{N}$.

El algoritmo es[5]

Datos: Un entero $n \geq 2$.
Salida: La lista de primos $\leq n$

1 $p = 2$;
2 Elimine los múltiplos de p (desde p^2 en adelante);
3 Sea $p =$primer entero, después de p, que no fue borrado. Repita el paso 2.;

E.6 Evaluando expresiones matemáticas escritas en lenguaje matemático común

Un evaluador de expresiones matemáticas o un "parser", es un programa que determina la estructura gramatical de una frase en un lenguaje. Este es el primer paso para determinar el significado de una frase tal como `"x^2+y"`, que para un lenguaje de programación significa traducirla en lenguaje de máquina.

En [9] encontrará un "parser" para evaluar expresiones matemáticas en Visual Basic. Sin embargo funciona bien con Excel. Además nos da la posibilidad de agregar funciones propias más complejas además de las que trae implementadas tales como `BesselJ(x,n)`, `HypGeom(x,a,b,c)` o `WAVE_RING()`. Por ejemplo, la función a trozos. Este parser fue desarrollado por Leonardo Volpi [9].

E.6.1 Usando clsMathParser. Sintaxis

Primero descargamos **`clsMathParser.zip`** (el adecuado según sea Excel 2003 o 2007).Actualmente existe este complemento en versión "Excel 97/2000/XP/2003" y versión "Excel2010and2007". Las instru-cciones y el complemento se puede descargar (agosto 2012) desde `http://www.thetropicalevents.com/Xnumbers60/`. La fuente original de Xnumbers (para Excel 97/2000/XP/2003) es `http://digilander.libero.it/foxes/`.


Software: Un cuaderno con **`clsMathparser`** Excel 2003.

En la carpeta **`clsMathParser`**, además de la documentación en pdf, vienen dos archivos: **`clsMathParser.cls`** y **`mMathSpecFun.bas`**. El primero es el parser y el segundo es una biblioteca con funciones especiales ya implementadas.

Para implementar un ejemplo de uso del parser, implementamos una hoja excel como la que se ve en la figura

[5]Una implementación eficiente puede ser encontrada en ([**?**])

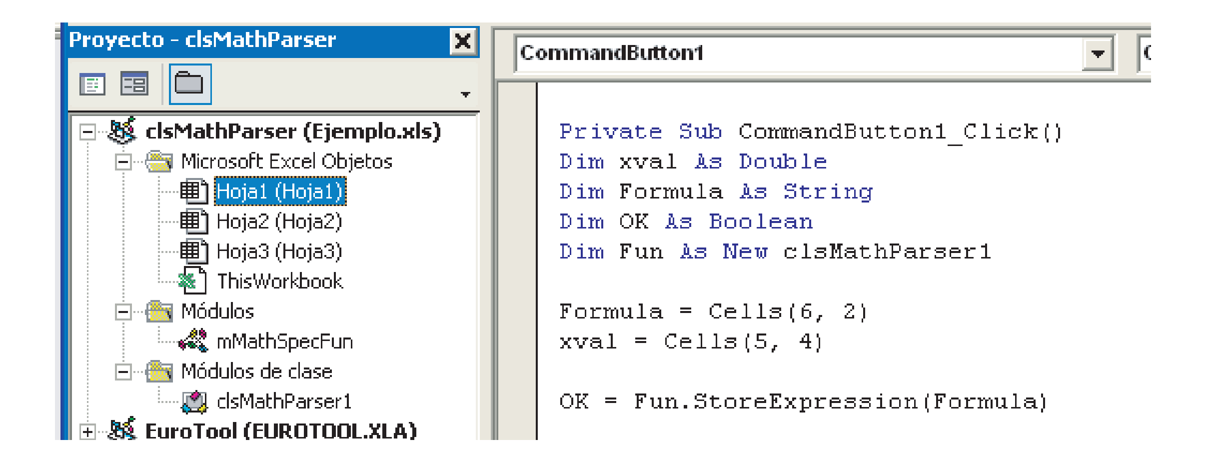
	A	B	C	D	E	F
1						
2	Usando clsMathParser (A Class for Math Expressions Evaluation in Visual Basic)					
3						
4						
5		Fórmula	Evaluar en x =	0		
6	f(x) =	cos(x^2)	f(0)=	1		

1. En el editor de VBA, seleccionamos la "Hoja1" y hacemos clic en el botón derecho del mouse. Ahí elegimos **`Importar Archivo...`**. Luego vamos a la carpeta **`clsMathParser`** y seleccionamos **`clsMathParser.cls`**

2. De la misma manera, importamos el archivo **`mMathSpecFun.bas`**.

   Comentario. Con esto ya podemos crear un objeto **`Fun`** con **`Dim Fun As New clsMathParser`** en el código del botón. Las evaluaciones se llevan a cabo con **`Fun.Eval1(xval)`**

3. En el editor VBA deberá quedar algo parecido a



   En este caso el módulo de clase quedó con el nombre **`clsMathParser`**. Así que el objeto **`Fun`** se construye con **`Dim Fun As New clsMathParser`**

4. Finalmente el código del botón es

Código VBA E.26:

```
Private Sub CommandButton1_Click()
    Dim xval As Double
    Dim Formula As String
    Dim FormulaEstaBien As Boolean
    Dim Fun As New clsMathParser rem  crea objeto Fun

    Formula = Cells(6, 2)
```

```
    xval = Cells(5, 4)

    FormulaEstaBien = Fun.StoreExpression(Formula) rem  leer rem Formularem  y analizar sintaxis
    If Not FormulaEstaBien Then
    msgbox("Error"+Fun.ErrorDescription)
    Exit Sub
    end If

    Cells(6, 3) = "f(" + str(valx) + ")= "
    Cells(6, 4) = Fun.Eval1(xval)   rem evaluar en la fórmula actual

End Sub
```

1. Las variables deben empezar con una letra, por ejemplo **`t, xi, time`**. Para evitar la ambigüedad, `xy` se interpreta como una variable. El producto requiere **`*`**, es decir **`x*y`** es el producto de x e y. Se permite multiplicación implícita solo para $x,y,$ y z, o sea, por ejemplo $2x$ se interpreta como **`2*x`**

2. Los argumentos de una función, si hay varios, se separan con una coma, por ejemplo **`max(x,y), root(x,y).`**

3. Las expresiones lógicas retornan 1 o 0. Como ejemplo de estas expresiones tenemos: **`(x<=1), (2<x<2)`**. Una expresión como "$x<0$ o $x>3$" se introduce como `(x<0)+(x>3)`

4. La expresión

$$f(x)=\begin{cases} x^2+x & \text{si} \quad x<0 \\ \log(x)+\sqrt{x} & \text{si} \quad x>2 \end{cases}$$

se introduce como `(x<0)*(x^2+x) + (x>2)*(log(x)+sqr(x))`

5. (Manejo de signos) Las siguientes expresiones *son equivalentes*

Código VBA E.27:

```
x^-n              x^(-n)

x*-sin(a)         x*(-sin(a))

-5*-2             -5*(-2)

-x^2              (-x)^2, o sea, -3^2 = (-3)^2 = 9
```

6. Constantes: π es **`pi#`** y e es **`e#`**. Por ejemplo

   (a) $\cos(x+\pi)+e^x$ se introduce como `cos(x+pi#)+e#^x`

(b) e^{-x^2} se debe introducir como `1/e#^(x^2)` o como `e#^(-1*x^2)` pues `e#^(-x^2)` $= e^{x^2}$

7. Constantes físicas.

Constante de Planck	h#	**6.6260755e-34 J s**
Constante de Boltzmann	K#	**1.380658e-23 J/K**
Carga elemental	q#	**1.60217733e-19 C**
Número de Avogadro	A#	**6.0221367e23 particles/mol**
Velocidad de la luz	c#	**2.99792458e8 m/s**
Permeabilidad del vacio (m)	mu#	**12.566370614e-7 T2m3/J**
Permitividad del vacio (e)	eps#	**8.854187817e-12 C2/Jm**
Masa del Electron	me#	**9.1093897e-31 kg**
Masa del Proton	mp#	**1.6726231e-27 kg**
Masa del Neutron	mn#	**n 1.6749286e-27 kg**
Constante Gas	R#	**8.31451 m2kg/s2k mol**
Constante gravitacional	G#	**6.672e-11 m3/kg s2**
Aceleración de la gravedad	g#	**9.80665 m/s2**

Las constantes físicas deben ir seguidas por sus unidades de medida: **"1␣s"**, **"200␣m"**, **"25␣kg"**, **"150␣MHz"**, **"0.15␣uF"**, **"3600␣kohm"**. Se utilizan de manera análoga a las constante matemáticas. Por ejemplo `eps# * S/d#`, `sqr(m*h*g#),s0+v*t+0.5*g#*t^2`

8. Símbolos y Funciones

```
+                 adición
-                 resta
*                 multiplicación
/                 division              35/4 = 8.75
%                 porcentaje            35% = 3.5
\                 división entera       35\4 = 8 ^            elevar a
potencia          3^1.8 = 7.22467405584208 (°) | |          valor
absoluto          |-5|=5 !           factorial              5!=120
abs(x)            valor absoluto        abs(-5)= 5 atn(x)
atan(x)           x en radianes
cos(x)            x en radianes
sin(x)            x en radianes
exp(x)            exponencial           exp(1) = 2.71828182845905, es decir, e^1
fix(x)            parte entera          fix(-3.8) = 3 int(x)
parte entera      int(-3.8) = -4 dec(x)
parte decimal     dec(-3.8) = -0.8 ln(x)        logaritmo natural      x>0 log(x)
log natural       logN(x,n)    logaritmo base N       logN(16,2) = 4
rnd(x)             random      retorna un valor aleatorio entre  x y 0
sgn(x)             signo       retorna 1 si x >0, 0 si x=0, -1 si x<0 sqr(x)
raíz cuadrada      sqr(2)=1.4142135623731, también 2^(1/2)
raíz cúbica        root(x,3)
raíz n-ésima       x^(1/n)
mod(a,b)     mod(29,6) = 5,  mod(-29 ,6) = -5 comb(n,k)
```

```
combinaciones  comb(6,3) = 20 , comb(6,6) = 1 perm(n,k)
permutaciones    perm(8,4) = 1680
```

También, entre otras:

```
min(a,b),max(a,b), sech(x), coth(x), acsch(x), asech(x), acoth(x),
round(x,d), mcd(a,b), mcm(a,b,gcd(a,b),lcm(a,b), csc(x), sec(x),
cot(x), acsc(x), asec(x), acot(x), csch(x),tan(x), acos(x),
asin(x), cosh(x), sinh(x), tanh(x), acosh(x), asinh(x), atanh(x),
and(a,b),or(a,b),not(a),xor(a,b), Psi(x), DNorm(), CNorm(),
DPoisson(),CPoisson(),DBinom(),CBinom(), Si(x) (SineIntegral),
Ci(x),FresnelS(x), etc.
```

9. Métodos

Código VBA E.28:

```
StoreExpression(f)         Almacena y revisa la sintaxis Eval(x)
Evalúa la expresión (que posiblemente tenga varias variables)
Eval1(x)                   Evalúa una expresión de una variable
```

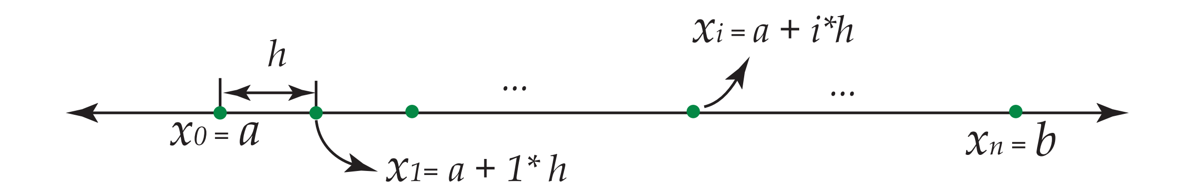
`Eval()` se usa cuando vamos a evaluar expresiones con varios parámetros y/o variables. **`Eval1()`** es para funciones de una sola variable.

Vamos a ver algunos ejemplos en los que se aplican estos métodos.

E.6.2 Ejemplo: un graficador 2D

Podemos implementar de manera muy sencilla un graficador de funciones $y = f(x)$.

Para esto, leemos la ecuación de la función $y = f(x)$ en una celda y el intervalo de graficación $[a,b]$. Para hacer el gráfico necesitamos una tabla de valores (x_i, y_i). Lo que hacemos es partir el intervalo $[a,b]$ con $n+1$ puntos x_i, $i = 0,1,...,n$; y evaluamos la función en cada uno de estos x_i. Si los puntos son equidistantes, la distancia de separación es $h = (b-a)/n$ y este caso



$x_0 = a + 0 \cdot h = a$

$x_1 = a + 1 \cdot h$

$x_2 = a + 2 \cdot h$

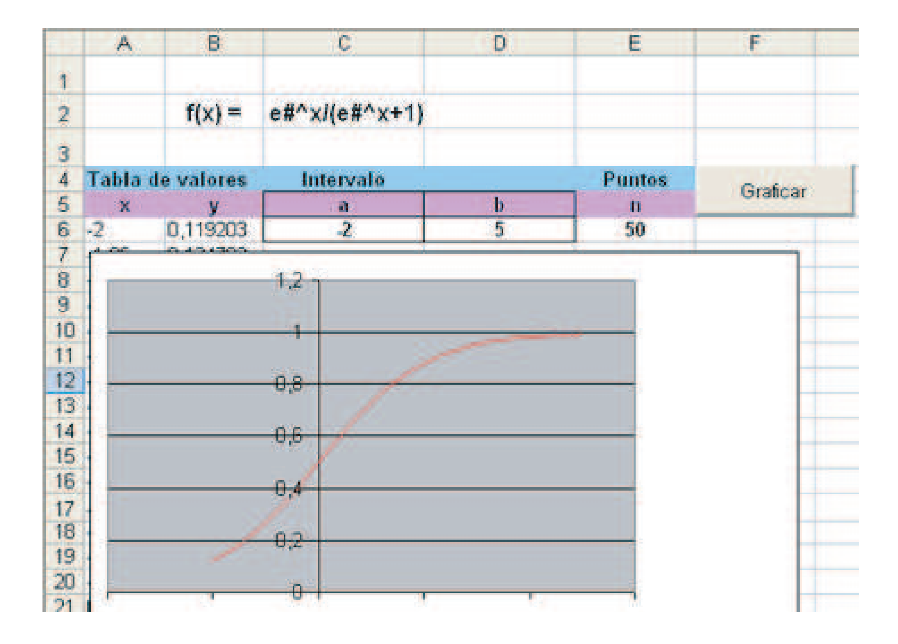
$\vdots$

$x_i = a + i \cdot h$

$x_n = a + n \cdot h = b$

Entonces la tabla de puntos es el conjunto $\{(a + i \cdot h, f(a + i \cdot h)),\; i = 0, 1, ..., n\}$

Supongamos que el graficador va a quedar en una hoja como la que se ve en la figura



Entonces la tabla de puntos se puede generar con el código

Código VBA E.29:

```
n = Cells(6, 5) a = Cells(6, 3) b = Cells(6, 4) h = (b - a) / n
Formula = Cells(2, 3)

FormulaEstaBien = Fun.StoreExpression(Formula)     rem lectura de la fórmula

If Not FormulaEstaBien Then
    msgbox("Error"+Fun.ErrorDescription)
    Exit Sub
end If

For i = 0 To n
    Cells(6 + i, 1) = a + i * h             rem xirem s
    Cells(6 + i, 2) = Fun.Eval1(a + i * h) rem yirem s
Next i
```

Luego debemos seleccionar la tabla de valores y hacer un gráfico (chart). Si hay un gráfico previo lo podemos borrar (que no es necesario si uno quiere comparar gráficos por ejemplo). Estas dos cosas se pueden lograr con el código

Código VBA E.30:

```
rem ----------------------- eliminar gráficos anteriores----------------
  Set chartsTemp = ActiveSheet.ChartObjects
  If chartsTemp.Count > 0 Then
     chartsTemp(chartsTemp.Count).Delete
  End If

rem ----------------------- gráfico------------------------------------
datos = Range(Cells(6, 1),
Cells(6 + n, 2)).Address rem rango a graficar Set graf = Charts.Add
rem gráfico

With graf              rem características
    .Name = "Gráfico"
    .ChartType = xlXYScatterSmoothNoMarkers  rem tipo de gráfico  XY(Dispersión)
    .SetSourceData Source:=Sheets("Hoja1").Range(datos), PlotBy:=xlColumns
    .Location Where:=xlLocationAsObject, Name:="Hoja1"
  End With
```

El código completo del botón que levanta el gráfico es

Código VBA E.31:

```
Private Sub CommandButton2_Click()
    Dim n As Integer
    Dim h As Double
    Dim Formula As String
    Dim graf   As Chart
    Dim chartsTemp As ChartObjects rem contador de charts (gráficos) para eliminar el anterior
    Dim FormulaEstaBien As Boolean
    Dim Fun As New clsMathParser

    n = Cells(6, 5)
    a = Cells(6, 3)
    b = Cells(6, 4)
    h = (b - a) / n
    Formula = Cells(2, 3)

    FormulaEstaBien = Fun.StoreExpression(Formula)    rem lectura de la fórmula
    If Not FormulaEstaBien Then
    msgbox("Error"+Fun.ErrorDescription)
    Exit Sub
    end If
```

```
    For i = 0 To n
    Cells(6 + i, 1) = a + i * h
    Cells(6 + i, 2) = Fun.Eval1(a + i * h)
    Next i


rem ----------------------- eliminar gráficos anteriores-------------
    Set chartsTemp = ActiveSheet.ChartObjects
    If chartsTemp.Count > 0 Then
        chartsTemp(chartsTemp.Count).Delete
     End If
rem -----------------------------------------------------------------


    datos = Range(Cells(6, 1), Cells(6 + n, 2)).Address rem rango a graficar
    Set graf = Charts.Add    rem gráfico y sus caraterísticas

    With graf
        .Name = "Gráfico"
        .ChartType = xlXYScatterSmoothNoMarkers
        .SetSourceData Source:=Sheets("Hoja1").Range(datos), PlotBy:=xlColumns
        .Location Where:=xlLocationAsObject, Name:="Hoja1"
     End With

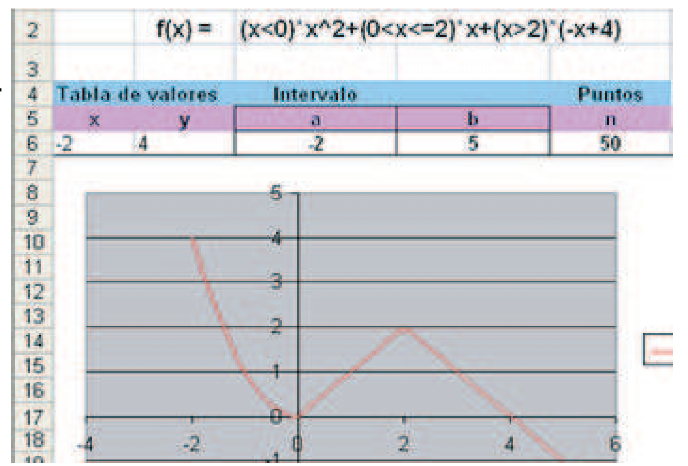
End Sub
```

Observe que se pueden graficar funciones a trozos. Por ejemplo, para graficar

$$f(x) = \begin{cases} x^2 & \text{si} \quad x \in]-\infty, 0[\\ x & \text{si} \quad x \in]0, 2] \\ 4-x & \text{si} \quad x \in]2, \infty[\end{cases}$$

digitamos la función como

f(x) =(x<0)*(x^2)+(0<x<=2)*x+(x>2)*(4-x)



EJERCICIOS

Implemente un graficador para curvas definidas por una ecuación parámetrica.

Nota. Algunas curvas, como los círculos, las elipses, etc., se definen por medio de ecuaciones parámetricas

$$(x(t), y(t)),\ \ t \in [a,b].$$

Por ejemplo

- Las ecuaciones paramétricas de un círculo de radio r y centro en (h,k) son

$$(h + r\cos(t), k + r\text{sen}(t)),\ \ t \in [0, 2\pi].$$

- Las ecuaciones paramétricas de un cicloide son $(a > 0)$

$$x = a(\theta - \text{sen}\theta),\ \ y = a(1 - \cos\theta),\ \ \theta \in \mathbb{R}.$$

- Una curva con ecuación en polares $r = f(\theta),\ \ \theta \in [a,b]$; tiene ecuación parámetrica

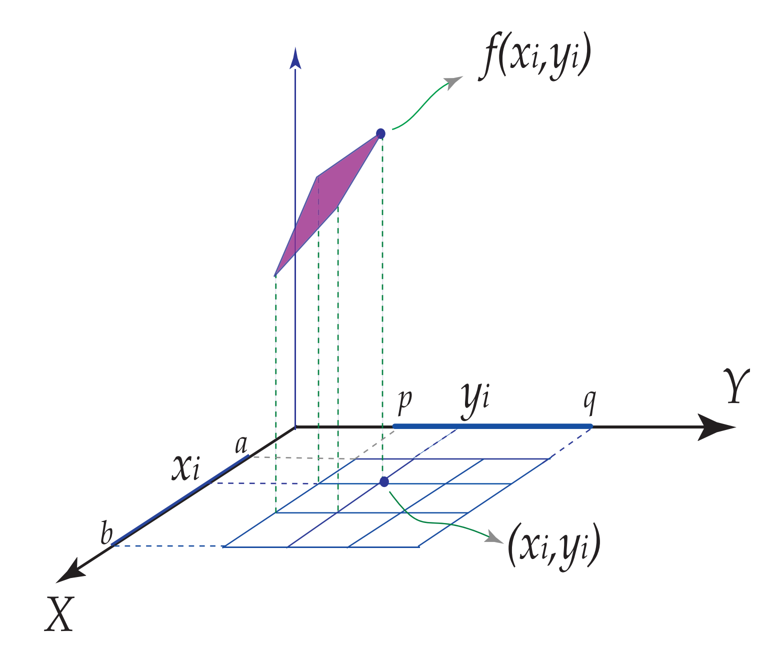
$$(f(\theta) \cdot \cos(\theta), f(\theta) \cdot \text{sen}(\theta)),\ \ t \in [a,b].$$

E.6.3 Ejemplo: un graficador de superficies 3D

Podemos implementar de manera muy sencilla un graficador de superficies de ecuación $z = f(x,y)$.
Para esto solo hay que observar que en vez de una tabla de puntos $(x_i, y_i, z_i(x_i, y_i))$ en tres columnas, Excel necesita una tabla de la forma

	x_0	x_1	$\cdots$	x_n
y_0	$f(x_0,y_0)$	$f(x_1,y_0)$	$\cdots$	$f(x_n,y_0)$
y_1	$f(x_0,y_1)$	$f(x_1,y_1)$		
$\vdots$				
y_n	$f(x_0,y_n)$	$f(x_1,y_n)$	$\cdots$	$f(x_n,y_n)$

Los intervalos de graficación son $x \in [xmin, xmax]$ y $y \in [ymin, ymax]$. Cada intervalo lo partimos en n puntos (que puede ser modificado por el usuario).



Si `hx = (xmax-xmin)/n` y si `{hy = (ymax-ymin)/n`, entonces

```
xi = xmin + i*hx,     i = 0,1,...,n

yj = ymin + j*hy,     j = 0,1,...,n
```

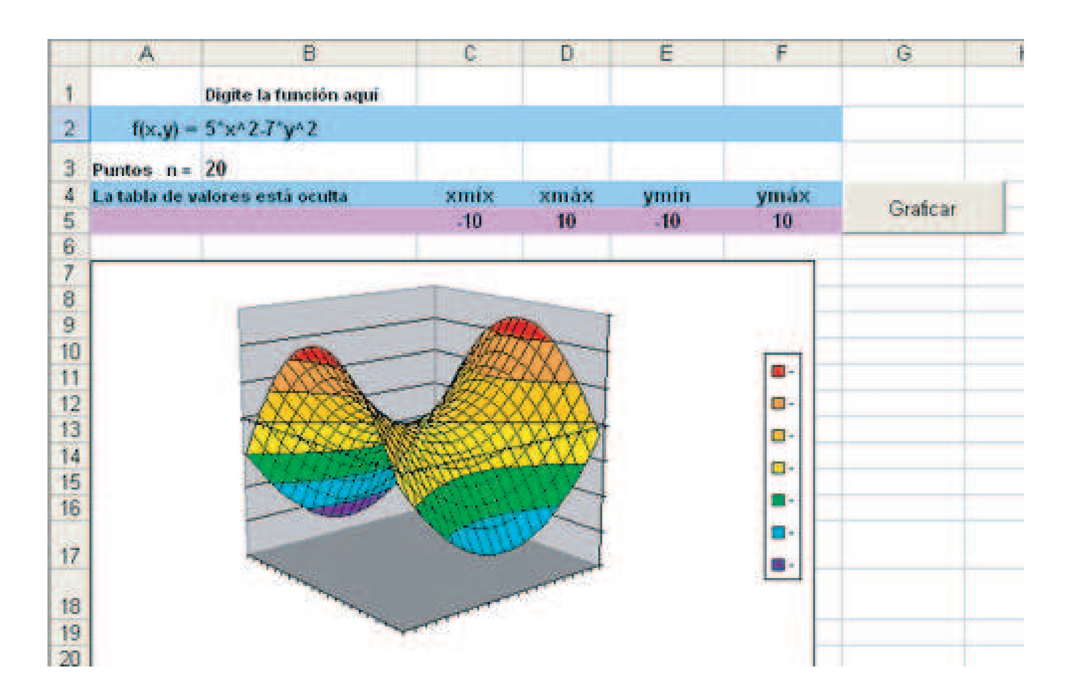
Para que la tabla quede en el formato adecuado, debemos usar dos ciclos iterativos anidados

Código VBA E.32:

```
For i = 0 To n
    xi = xmin + i * hx
    Cells(7, 2 + i) = xi        rem fila de las xi
    For j = 0 To n
        yi = ymin + j * hy
        Cells(8 + j, 1) = yi    rem columna de la yi

        OK = Fun.StoreExpression(fxy)      rem formula actual es rem f(x,y)rem
        If Not OK Then GoTo Error_Handler
        Fun.Variable("x") = xi             rem  asignación para evaluar
        Fun.Variable("y") = yi
        Cells(8 + j, 2 + i) = Fun.Eval()   rem retorna f(xi,yi)
        Next j
Next i
```

Una vez que se genera la tabla de datos, la seleccionamos y la ocultamos (porque es muy grande y llena de decimales). Luego generamos el gráfico 3D.



El código completo es

Código VBA E.33:

```
Private Sub CommandButton2_Click()
    Dim xmin, xmax, ymin, ymax, hx, hy, xi, yi As Double
    Dim n As Integer
    Dim fxy As String rem función f(x,y)
    Dim graf   As Chart
    Dim FormulaEstaBien As Boolean
    Dim Fun As New clsMathParser      rem  así se llama el módulo de clase aquí

    fxy = Cells(2, 2)
    xmin = Cells(5, 3)
    xmax = Cells(5, 4)
    ymin = Cells(5, 5)
    ymax = Cells(5, 6)
    n = Cells(3, 2) rem  número de puntos n x n
    hx = (xmax - xmin) / n
    hy = (ymax - ymin) / n

    If hx > 0 And hy > 0 And n > 0 Then
        For i = 0 To n
            xi = xmin + i * hx
            Cells(7, 2 + i) = xi
            For j = 0 To n
                yi = ymin + j * hy
                Cells(8 + j, 1) = yi
```

```
            FormulaEstaBien = Fun.StoreExpression(fxy)     rem fórmula actual es rem f(x,y)rem
            If Not FormulaEstaBien Then
               msgbox("Error"+Fun.ErrorDescription)
               Exit Sub
            end If

            Fun.Variable("x") = xi
            Fun.Variable("y") = yi
            Cells(8 + j, 2 + i) = Fun.Eval() rem retorna f(xi,yi)
        Next j
    Next i

    End If
    rem ------------------------ eliminar gráficos anteriores-------------
    Set chartsTemp = ActiveSheet.ChartObjects
    If chartsTemp.Count > 0 Then
        chartsTemp(chartsTemp.Count).Delete
    End If
    rem -------------------------------------------------------------
    datos = Range(Cells(7, 1), Cells(7 + n, n + 2)).Address rem rango a graficar
    Range(datos).Select
    Selection.NumberFormat = ";;;" rem ocular celdas
    Charts.Add
    ActiveChart.ChartType = xlSurface
    ActiveChart.SetSourceData Source:=Sheets("Hoja1").Range(datos), PlotBy:=xlColumns
    ActiveChart.Location Where:=xlLocationAsObject, Name:="Hoja1"

End Sub
```

Nota: para rotar la figura, seleccione el área del gráfico y haga clic con el botón derecho del mouse, seleccione en el menú la opción Vista en 3D...

EJERCICIOS

1. Implemente un graficador para curvas y/o superficies con ecuación paramétrica.

series numéricas y series de potencias. Implementaremos una función que calcula la suma parcial $S_N = \sum_{k=k_0}^{N} f(k)$.

Luego la aplicaremos a la serie $\sum_{k=1}^{\infty} \frac{2^k}{k!}$.

C	D	E	F	G	H
				Suma parcial	
				Sumar	
	10				
	$\sum$	2^k/k!	=	6,388994709	
k =	1				

Para esto vamos a definir una función

$$\texttt{SumaParcial(formula, K_0, N)} = \sum_{k=K_0}^{N} f(k).$$

Esta función revisa la sintaxis de la fórmula y calcula la suma parcial.

Código VBA E.34:

```
Function SumaParcial(formula, Kini, N)
    Dim suma As Double
    Dim FormulaEstaBien As Boolean
    Dim Fun As New clsMathParser         rem  así se llama el módulo de clase aquí
    FormulaEstaBien = Fun.StoreExpression(formula)  rem  fórmula actual es "formula"

    If Not FormulaEstaBien Then
    msgbox("Error"+Fun.ErrorDescription)
    Exit Sub
    end If

    suma = 0
    For i = Kini To N
        suma = suma + Fun.Eval1(i) rem evalúa en "formula"
    Next i
SumaParcial = suma End Function
```

Ahora, podemos aplicar esta función con un botón

Código VBA E.35:

```
Private Sub CommandButton1_Click()
Dim formula1 As String
Dim N
Dim kk As Integer

formula1 = Cells(8, 5)
```

```
N = Cells(7, 4)
kk = Cells(10, 4)
Cells(8,7) = SumaParcial(formula1, kk, N)
End Sub
```

Dada una serie numérica $\sum_{k=k_0}^{\infty} f(k)$, vamos a desplegar las primeras N sumas parciales $S_N = \sum_{k=k_0}^{N} f(k)$, de la celda B4 en adelante.

	A	B	C	D
1	Sumas parciales de la serie			
2		Fórmula f(k)	Sumas Parciales	N
3	SN	2^k/k!		20
4	S 1 =	2		
5	S 2 =	4		
6	S 3 =	5,333333333		
7	S 4 =	6		
8	S 5 =	6,266666667		
9	S 6 =	6,355555556		

Usando la función **`SumaParcial(formula, Kini, N)`**, solo debemos calcular cada suma parcial por separado.

Código VBA E.36:

```
For i = 1 to N
    Cells(3+i,2)= SumaParcial(formula,Kini,i)
Next i
```

El código del botón es

Código VBA E.37:

```
Private Sub CommandButton1_Click()
    Dim x0 As Double
    Dim formula1 As String
    Dim N As Integer

    formula1 = Cells(3, 2)
    N = Cells(3, 4)

    For i = 1 To N   rem sumas parciales S1, S2, ... SN
        Cells(3 + i, 1) = "S" + str(i) + " = "
        Cells(3 + i, 2) = SumaParcial(formula1, 1, i)
    Next i
End Sub
```

Usando Eval(). Ahora vamos a cambiar a series de potencias. Dada una serie de potencias $\sum_{k=k_0}^{\infty} f(x,k)$, implementaremos una subrutina que calcula la suma parcial $S_N = \sum_{k=k_0}^{N} f(x_0,k)$. Luego la aplicaremos a la serie $\sum_{k=1}^{\infty} \frac{x^k}{k!}$, con $x = 2$.

	C	D	E	F	G	H
5					Suma parcial	
6					Sumar	x
7		5				2
8		$\sum$	x^ k/k!	=	6,266666667	
9						
10	k =	1				

Vamos a implementar una nueva función **`SumaParcial(laformula, Kini, N, xx)`**.

Aquí la situación es diferente, $f(x,k)$ tiene dos variables, por tanto no podemos usar **`Eval1()`** porque este método es para funciones de una sola variable.

`Eval()` evalúa la expresión actual almacenada y retorna su valor. Para evaluar, se debe especificar el valor de cada variable con **`Fun.Variable()`**. En nuestro caso debemos hacer la especificación **`Fun.Variable("x")=xx`** y **`Fun.Variable("k")=i`**, antes de llamar a **`Eval()`**. Veamos el código de la función **`SumaParcial(formula, Kini, N, xx)`**

Código VBA E.38:

```
Function SumaParcial(formula, Kini, N, xx)
    Dim suma As Double
    Dim FormulaEstaBien As Boolean
    Dim Fun As New clsMathParser        rem  así se llama el módulo de clase aquí
    FormulaEstaBien = Fun.StoreExpression(formula)  rem  fórmula actual es "laformula"

    If Not FormulaEstaBien Then
        msgbox("Error"+Fun.ErrorDescription)
        Exit Sub
    end If

    suma = 0
    Fun.Variable("x") = xx
    For i = Kini To N
        Fun.Variable("k") = i
        suma = suma + Fun.Eval() rem evalúa en "formula" actual
SumaParcial = suma End Function
```

Ahora, el código del botón sería

Código VBA E.39:

```
Private Sub CommandButton1_Click()
    Dim formula1 As String
    Dim N, kk As Integer
```

```
    Dim x0 As Double

    formula1 = Cells(8, 5)
    N = Cells(7, 4)
    kk = Cells(10, 4)
    x0 = Cells(7, 8)
    Cells(8, 7) = SumaParcial(formula1, kk, N, x0)
End Sub
```

EJERCICIOS

1. Implemente un función para series alternadas que además de dar la suma parcial pedida, indique la estimación del error. Recuerde que si $\sum_{k=1}^{\infty}(-1)^k f(k)$ es alternada entonces si el error cometido al aproximar la serie con la suma parcial $S_N = \sum_{k=1}^{N}(-1)^k f(k)$ es R_N, entonces $|R_N| \leq a_{N+1}$.

Nota: Puede ser importante conocer la primera o la última celda de una selección. Para esto podemos usar la propiedad **`Address`**

Código VBA E.40:

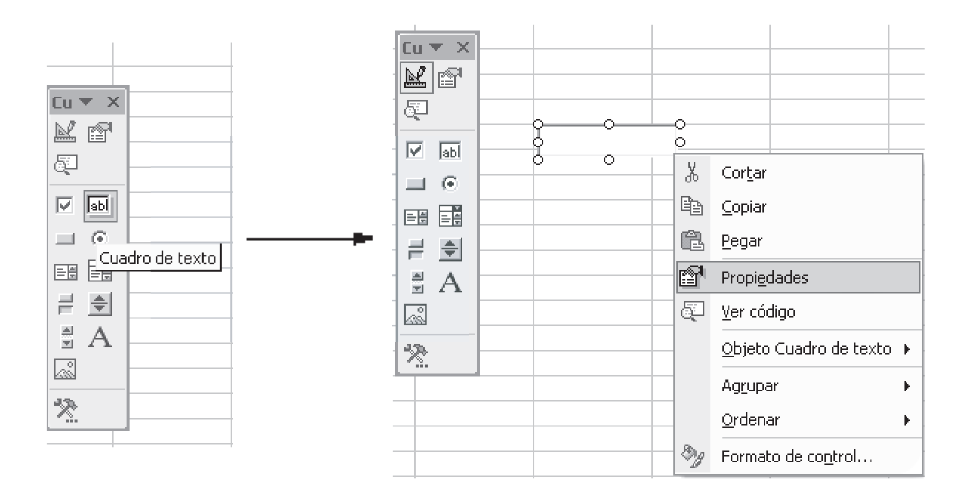
```
  Set R = Selection
    adr = R(1, 1).Address  rem primera celda de la selección, por ejemplo A1
    num = Right(adr, 1)    rem primera posición a la derecha de adr (1 en este caso)
```

Código VBA E.41:

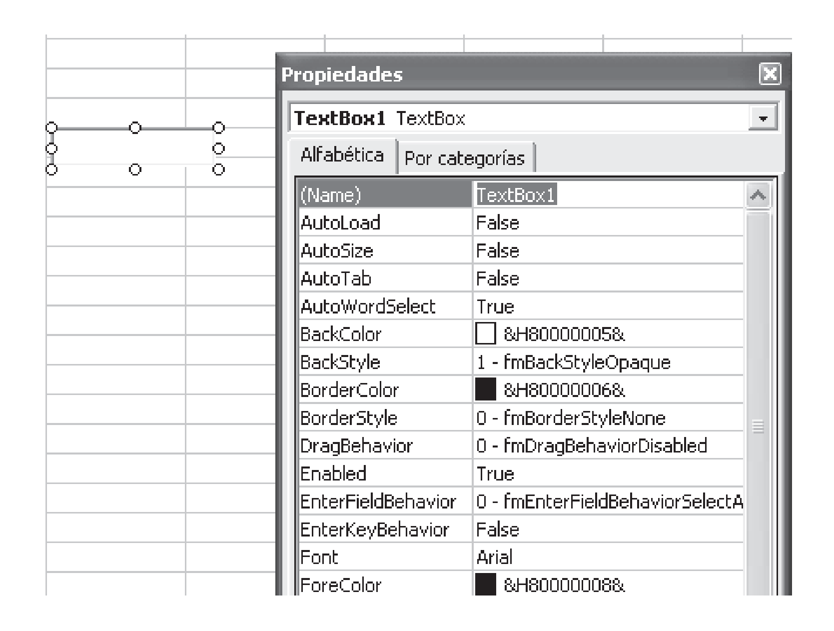
```
If m mod n = 0 Then  rem si m es múltiplo de n .... End If
```

E.6.4 Campos de Texto.

A veces puede ser cómodo entrar datos en un campo de texto (TextBox) en vez de una celda. Para incluir un campo de texto en una hoja se procede de manera análoga a como se incluye un botón. Primero, en la barra de controles se selecciona el campo de texto y se da clic en la hoja



Luego, se tiene acceso a las propiedades dando clic derecho en el campo de texto. Observe que el primer campo de texto tiene nombre (Name) “TextBox1". Este nombre se usa para identificar este campo de texto. Este nombre se puede cambiar, pero no es necesario.



En el código VBA, se puede leer lo que el usuario escribió en el campo de texto con la instrucción

```
Dim Formula As String
Formula = TextBox1.Text
```

en este caso, el contenido del campo de texto se almacena en la variable “Formula".

Vamos a implementar de nuevo un “evaluadorrem rem de funciones. La hoja Excel luce así

	A	B	C	D	E	F	G
1							
2							
3	Digite	Campo de Texto			Celda		
4	f(x) =	x^2-cos(x)-1		x =	34	Evaluar f	
6							
7							
8				f(34) =	1155,84857		

El código es el siguiente

Código VBA E.42:

```
Private Sub CommandButton1_Click() evaluarEimprimirFun End Sub


Sub evaluarEimprimirFun()
    Dim xvalor As Double
    Dim Formula As String
    Dim estaBien As Boolean
    Dim Fun As New clsMathParser rem  crear objeto Fun

    Formula = TextBox1.Text
    xvalor = Cells(4, 5)

    rem  leer rem Formularem  y analizar sintaxis
    FormulaEstaBien = Fun.StoreExpression(Formula)
    If Not FormulaEstaBien Then
    msgbox("Error"+Fun.ErrorDescription)
    Exit Sub
end If

    Cells(8, 4) = "f(" + str(xvalor) + ") = "
    rem evaluar en la fórmula actual
    Cells(8, 5) = Fun.Eval1(xvalor)

End Sub
```

Nota: Observe que en este ejemplo, como usamos el evaluador de fuciones **`clsMathParser.cls`**, en el campo de texto de la función, los números con decimales usan punto. En cambio en las celdas, si necesitamos decimales, debemos atender a la configuración regional del computador (coma o punto).

Bibliografía

[1] W. Gautschi. *Numerical Analysis. An Introduction.* Birkhäuser, 1997.

[2] P. Henrici.*Essentials of Numerical Analysis.* Wiley, New York, 1982.

[3] J. Stoer,. *Introduction to Numerical Analysis.* 3rd ed. Springer, 2002.

[4] D. Kahaner, K. Moler, S. Nash. *Numerical Methods and Software.* Prentice Hall.1989.

[5] D. Kincaid, W. Cheney. *Numerical Analysis. Mathematics of Scientific Computing.* Brooks-Cole Publishing Co.1991.

[6] R. Burden, J. Faires. *AnAlisis NumArico.* 6ta ed. Thomson. 1998.

[7] E. Cheney, *Introduction to Approximation Theory.* Internat. Ser. Pure ans Applied Mathematics. McGraw-Hill. 1966.

[8] J.P. Berrut, L. N. Trefethen. "Barycentric Lagrange Interpolation" Siam Rewiew. Vol. 46, No. 3. 2004.

[9] J. Higham, "The numerical stability of barycentric Lagrange interpolation". IMA Journal of Numerical Analysis 24. 2004.

[10] Kaufman, E. y Lenker, T. "Linear Convergence and the Bisection Algorithm". Amer. Math. Monthly, 93:48-51, 1986.

[11] Brent, R.P. "An Algorithm with Guaranteed Convergence for Finding a Zero of a Function." Computer Journal 14 (1971), 422-425. `http://web.comlab.ox.ac.uk/oucl/work/richard.brent/pd/rpb005.pdf`

[12] Press, W.; Teukolsky, S.; Vterling, W.; Flannery, B. *Numerical Recipes in C. The Art of Scientific Computing.* 2da Ed. Cambridge Universite Press. 1992.

[13] Chapra, S.; Canale, R. *Métodos Numéricos para Ingenieros.* Ed. Mc Graw Hill, 4a. ed., 2002.

[14] Mathews, J; Fink, K.*Métodos Numéricos con MATLAB*. Prentice Hall, 3a. ed., 2000.

[15] Dahlquist, G. Björk, A. *Numerical Mathematics in Scientific Computation.* `www.mai.liu.se/~akbjo`. Consultada en Mayo, 2006.

[16] Zeng, Z. "MultRoot - A Matlab package computing polynomial roots and multiplicities". `http://www.neiu.edu/~zzeng/Papers/zrootpak.pdf)`. Consultada en Mayo, 2006.

[17] Zeng, Z. "Computing multiple roots of inexact polynomials". `http://www.neiu.edu/~zzeng/Papers/zroot.ps`. Consultada en Mayo, 2006.

[18] Plofker, Kim. "An example of the secant method of iterative approximation in a fifteenth-century Sanskrit text", *Historia Math.* 23, 246-256.

[19] DAez, Pedro. "A note on the convergence of the secant method for simple and multiple roots" Applied Mathematics Letters, Vol. 16, Issue 8, pp. 1211-1215, 200.

[20] Ralston, A.; Rabinowitz, P. *A First Course in Numerical Analysis.* 2nd ed. Dover, 1978.

[21] Demidovich, B.; Maron, I. *CAlculo NumArico Fundamental*. 2da ed. Paraninfo (Madrid), 1985.

[22] Scarborough, J. *Numerical Mathematical Analysis.* 3rd ed. Oxford University Press, 1955.

[23] Moler, K. *Numerical Computing with MATLAB.* `http://www.mathworks.com/moler/` Consultada en Enero, 2006.

[24] Traub, J. F. *Iterative Methods for the Solution of Equations.* AMS Bookstore, 1982. Disponible en google book search: `http://books.google.com/`

[25] Brent, R.P. "An Algorithm with Guaranteed Convergence for Finding a Zero of a Function." Computer Journal 14 (1971), 422-425. `http://web.comlab.ox.ac.uk/oucl/work/richard.brent/pd/rpb005.pdf`

[26] A. Jeffrey, Hui-Hui Dai. *Handbook of Mathematical Formulas and Integrals.* 4th edition. Academic Press. 2008.

[27] A. Channelle. *Beginning OpenOffice 3. From Novice To Professional.* Apress. 2009.

[28] A. Pitonyak. *OpenOffice.org Macros Explained*. Hentzenwerke Publishing. 2004.

[29] L. Godart, Bernard Marcelly. *Programmation OpenOffice.org 2. Macros OOo Basic et API.* Editions Eyrolles. 2006.

[30] M.Baeza S. *Aprendiendo OOo Basic*. En `http://www.universolibre.org/node/1`

[31] M. A. Bain. *Learn OpenOffice.org. Spreadsheet Macro Programming OOo Basic and Calc Automation.* Packt Publishing. 2006.

[32] R. Graham, D. Knuth y O.Patashnik. *Concrete Mathematics: A Foundation for Computer Science*, 2nd ed. Addison-Wesley, pp. 272-282, 1994.

[33] OOo Forum.org. ”Solved:Sheet Cell Array Functions in Basic.“ `http://www.oooforum.org/forum/viewtopic.phtmlAt=23520`. Consultada el 18 de agosto 2010.

[34] OOo Forum.org. ”How to use callFunction()A.“ `http://www.oooforum.org/forum/viewtopic.phtmlAp=389084#389084`. Consultada el 24 de agosto 2010.

[35] OpenOffice.org Wiki. “OpenOffice.org BASIC Programming Guide.” En `http://wiki.services.openoffice.org/wiki/Documentation/BASIC_Guide`

Solución de los Ejercicios

Soluciones del Capítulo 1

1.1 $|p-\tilde{p}| \leq 0.5 \times 10^{-4}$ pero $|p-\tilde{p}| \not< 0.5 \times 10^{-5}$ así que tenemos 4 deciamles correctos. Luego, solo 0.001 y 0.0002 son $> 10^{-4}$ así que solo hay dos cifras significativas. solo

1.2 $|\ p-\tilde{p}\ | \leq 0.000001 \leq 0.5 \times 10^{-5}$ pero no se sabe si $|\ p-\tilde{p}\ | < 10^{-6}$, entonces solo podemos asegurar 5 decimales correctos.

1.3 Sirve cualquier $\delta \leq 0.5 \times 10^{-3}$.

1.9 Usando tres decimales todo el tiempo, obtenemos $x = 0.002$ y $y = 0.384$ mientras que la solución correcta es $x = 1$ y $y = 1$.

Soluciones del Capítulo 2

2.1.a

$$\begin{aligned} P_3(x) &= \frac{1\cdot(x-1)(x-3)(x-4)}{(0-1)(0-3)(0-4)} \\ &+ \frac{2\cdot(x-0)(x-3)(x-4)}{(1-0)(1-3)(1-4)} \\ &+ \frac{0\cdot(x-0)(x-1)(x-4)}{(3-0)(3-1)(3-4)} \\ &+ \frac{4\cdot(x-0)(x-1)(x-3)}{(4-0)(4-1)(4-3)} \end{aligned}$$

2.1.c $f(3.5) \approx P_3(3.5) = 1.21875$

2.2.a $P_3(x) = x(x-1)(x-3)(x-4)\left(\frac{1}{3(x-1)} - \frac{1}{12x} + \frac{1}{3(x-4)}\right)$

2.2.b $P_3(x) = \dfrac{\frac{1}{3(x-1)} - \frac{1}{12x} + \frac{1}{3(x-4)}}{-\frac{1}{6(x-3)} + \frac{1}{6(x-1)} - \frac{1}{12x} + \frac{1}{12(x-4)}}$

2.2.d $f(3.5) \approx P_3(3.5) = 1.21875$

2.3 Forma Modificada,

$$P_3(x) = (x-0.5)(x-0.4)(x-0.3)(x-0.2)\left(-\frac{4700.}{x-0.4} + \frac{2650.}{x-0.3} - \frac{200.}{x-0.2} + \frac{1750.}{x-0.5}\right)$$

Forma Baricéntrica,

$$P_3(x) = \frac{-\frac{4700.}{x-0.4} + \frac{2650.}{x-0.3} - \frac{200.}{x-0.2} + \frac{1750.}{x-0.5}}{-\frac{500.}{x-0.4} + \frac{500.}{x-0.3} - \frac{166.667}{x-0.2} + \frac{166.667}{x-0.5}}$$

$f(0.35) \approx P_3(0.35) = 7.5375$

2.4

$$\begin{aligned} P_1(x) &= y_0 L_{n,0}(x) + y_1 L_{n,1}(x) \\ &= y_0\frac{(x-x_1)}{(x_0-x_1)} + y_1\frac{(x-x_0)}{(x_1-x_0)} \\ &= \frac{y_0(x-x_1) - y_1 x + y_1 x_0 + x_1 y_1 - x_1 y_1}{x_0 - x_1} \\ &= \frac{(y_0-y_1)}{(x_0-x_1)}(x-x_1) + y_1 \end{aligned}$$

2.5.a $P_2(x) = 44.875(x-0.4)(x-0.2) + 38.5x(x-0.2) - 77.75(x-0.4)x$

2.5.b $J_0(0.25) \approx P_3(0.25)/\pi = 0.974128$

2.6 La tabla acumulada es,

Salarios ($)	1000	2000	3000	4000
Frecuencia	9	39	74	116

La cantidad estimada de personas con salario entre $1000 y $1500 es $P_3(15) - 9 = 23.5 - 9 = 14.5$, es decir, unas 15 personas.

2.7 Como $1 + 3x^* = 1.75 \Longrightarrow x^* = 0.25$. El polinomio interpolante es,

$$P_2(x) = 9.72544\left(x - \frac{1}{3}\right)\left(x - \frac{1}{6}\right) - 7.49065x\left(x - \frac{1}{6}\right) - 2.54653\left(x - \frac{1}{3}\right)x$$

$\cos(1.75) \approx P_2(0.25) = -0.17054$

2.8.a Usamos la subtabla,

$v\ (m^3/kg)$	0.10377	0.11144
$s\ (kJ/Kg \cdot K)$	6.4147	6.5453

El polinomio interpolantes es $P_1(x) = -836.336(x - 0.11144) + 853.364(x - 0.10377)$. La entropía s para un volumen especifico v de $0.108m^3/kg$ es $6.48673(kJ/Kg \cdot K)$

2.8.b En este caso usamos los tres datos, el polinomio interpolantes es $P_2(x) = 38665.6(x - 0.1254)(x - 0.11144) + 22408.7(x - 0.10377)(x - 0.11144) - 61129.2(x - 0.1254)(x - 0.10377)$. La entropía s para un volumen especifico v de $0.108m^3/kg$ es $6.48753(kJ/Kg \cdot K)$

2.9 Usando toda la tabla obtenemos $f(0.25) \approx P_5(0.25) = 0.00120042$.

2.10 Como $P(x)$ es un polinomio, coincide con el polinomio interpolante, i.e., $P(x) = P_5(x)$. En la tabla aparecen los datos y la matriz de diferencias divididas,

x_i	y_i					
-1	3					
0	0	-3				
1	4	4	$\frac{7}{2}$			
2	0	-4	-4	$-\frac{5}{2}$		
3	1	1	$\frac{5}{2}$	$\frac{13}{6}$	$\frac{7}{6}$	
4	0	-1	-1	$-\frac{7}{6}$	$-\frac{5}{6}$	$-\frac{2}{5}$

$$\begin{aligned} P(x) = \; & 3 - 3(x+1) + \tfrac{7}{2}x(x+1) - \tfrac{5}{2}(x-1)x(x+1) + \tfrac{7}{6}(x-2)(x-1)x(x+1) \\ & -\tfrac{2}{5}(x-3)(x-2)(x-1)x(x+1) = -\frac{2x^5}{5} + \frac{19x^4}{6} - \frac{41x^3}{6} + \frac{x^2}{3} + \frac{116x}{15}. \end{aligned}$$

2.11 $f[x_2, x_3, x_4]$ es el coeficiente principal del polinomio que interpola los datos $(0.2, 2), (0.3, 3), (0.4, 4)$. Calculando la matriz de diferencias divididas (u observando que lo datos están sobre una recta) obtenemos que $f[x_2, x_3, x_4] = 0$

2.12 Usando la fórmula de diferencias divididas,

$$\begin{aligned} f[x_0, x_1, x_2] &= \frac{1}{x_2 - x_0}\left[\frac{y_2 - y_1}{x_2 - x_0} - \frac{y_1 - y_0}{x_1 - x_0}\right] \\ &= \frac{y_0}{(x_0 - x_1)(x_0 - x_2)} + \frac{y_1}{(x_1 - x_0)(x_1 - x_2)} + \frac{y_2}{(x_2 - x_0)(x_2 - x_1)}. \end{aligned}$$

2.13.a En la tabla aparecen los datos y la matriz de diferencias divididas,

x_i	y_i			
0	1			
1	2	1		
3	0	-1	$-\frac{2}{3}$	
4	4	4	$\frac{5}{3}$	$\frac{7}{12}$

La forma de Newton del polinomio interpolante es

$P_3(x) = 1 + x - \frac{2}{3}(x-1)x + \frac{7}{12}(x-3)(x-1)x.$

2.13.c $f(3.5) \approx P_3(3.5) = 1.21875.$

2.14 La matriz de diferencias divididas es,

$\frac{5}{4}$				
0.308	-0.00471			
0.132	-0.00088	0.0000127667		
0.079	-0.000265	2.05×10^{-6}	-2.67916×10^{-8}	
0.044	-0.000175	$3. \times 10^{-7}$	-4.37499×10^{-9}	4.48333×10^{-11}

El segundo coeficiente virial (estimado) a $450K$ es $13.8844(cm^3/mol)$

2.15 La matriz de diferencias divididas es,

-0.23			
-1.04	-0.054		
-0.57	0.0313333	0.00341333	
-0.64	-0.00466667	-0.0018	-0.000173778

El polinomio es $P_4(x) = 988 - 0.23(x-50) - 0.054(x-60)(x-50) + 0.00341333(x-65)(x-60)(x-50) - 0.000173778(x-75)(x-65)(x-60)(x-50)$.

A una temperatura de 68 grados Celsius, la densidad estimada es $978.084(kg/m^3)$

2.16 $f[x_i, x_j] = \dfrac{y_i - y_j}{x_i - x_j} = \dfrac{-(y_j - y_i)}{-(x_j - x_i)} = f[x_j, x_i]$

2.17 El polinomio es $P(x) = y_6 + f[x_6, x_7](x - x_6)$, es decir, $P(x) = y_6 + \dfrac{y_7 - y_6}{x_7 - x_6}(x - x_6)$.

2.18.a En la tabla aparecen los datos $(x_i, J_0(x_i))$ y la matriz de diferencias divididas,

x_i	$J_0(x_i)$		
0	1.14273		
0.2	0.989944	-0.763944	
0.4	0.980394	-0.0477465	1.79049

$P_2(x) = 1.14273 - 0.763944x + 1.79049(x-0.2)x.$

2.18.b $J_0(0.25) \approx P_2(0.25) = 0.974128$

2.19.a Para hacer la estimación necesitamos una tabla con las frecuencias acumuladas,

x	40	50	60	70	80
y	35	83	153	193	215

La cantidad de estudiantes con nota mayor o igual a 65. es $215 - P_4(65)$.

2.19.b La cantidad de estudiantes en el rango $55 - 65$ es $P_4(65) - P_4(55) = 57.5$, es decir, unos 58 estudiantes.

2.20 El peso aproximado de los bebés entre los 5 y 5.6 meses de vida es 15.67 libras.

2.21 En la tabla aparecen los datos $(x_i, J_0(x_i))$ y la matriz de diferencias divididas,

x_i	$\cos(1+3x_i)$		
0	0.540302		
0.166667	0.070737	-2.81739	
0.333333	-0.416147	-2.9213	-0.311742

$\cos(1.75) \approx P_2(0.25) = -0.17054.$

2.22.a La entropía s para un volumen especifico v de $0.108 m^3/kg$ es $6.48673 (kJ/Kg \cdot K)$.

2.22.b La entropía s para un volumen especifico v de $0.108 m^3/kg$ es $s(0.108 m^3/kg) = 6.48753 (kJ/Kg \cdot K)$.

2.23

x_i	y_i				
0	2				
1	3	1			
2	2	-1	-1		
3	1	-1	0	0,333333333	
4	3	2	1,5	0,5	0,041666667

2.24

$$|f(2.71) - P(2.71)| \leq \left| \frac{M}{3!}(2.71-1)(2.71-2)(2.71-3) \right|$$

$f^{(3)}(x) = 2/x$ y $f^{(4)}(x) = -2/x^2$. Como $f^{(4)}$ no se anula, $M = \text{máx}\{|f^{(3)}(1)|, |f^{(3)}(3)|\} = 2$.

$$|f(2.71) - P(2.71)| \leq 0.117363$$

2.25

$$|f(1.22) - P(1.22)| \leq \left| \frac{M}{4!}(1.22-0.5)(1.22-1.1)(1.22-1.2)(1.22-1.3) \right|$$

$f^{(4)}(x) = -\dfrac{48\left(4x^4 - 12x^2 + 1\right)}{\left(2x^2+1\right)^4}$ y $f^{(5)}(x) = \dfrac{384\left(4x^5 - 20x^3 + 5x\right)}{\left(2x^2+1\right)^5}$.

Los puntos críticos en $[0.5, 1.3]$ son $x = 0.513743, x = 0$. Entonces $M = 48$ y

$$|f(1.22) - P(1.22)| \leq 0.00027648$$

2.26

$$|e^{0.6} - P(0.6)| \leq \left| \frac{M}{3!}(0.6-0)(0.6-0.5)(0.6-1) \right|$$

$f^{(3)}(x) = f^{(4)}(x) = e^x$. No hay puntos críticos y $M = e$.

$$|e^{0.6} - P(0.6)| \leq 0.0108731$$

2.27 Aquí se pide estimar el error al interpolar $f(0.71) = \cos(3 \cdot 0.71 + 1)$ con $P(0.71)$. Si M es el máximo absoluto de $|f'''|$ en $[0,1]$, entonces,

$$|f(0.71) - P(0.71)| \leq \left| \frac{M}{3!}(0.71-0)(0.71-0.5)(0.71-1) \right|$$

Máximo absoluto de $|f'''|$

Puntos críticos: $f^{(4)}(x) = 0 \implies 81\cos(1+3x) = 0 \implies x = \dfrac{(2k+1)\frac{\pi}{2} - 1}{3}$, $k \in \mathbb{Z}$. Algunas soluciones son

$\{..., -2.95133, -1.90413, -0.856932, 0.190265, 1.23746, 2.28466, 3.33186, ...\}$.

El único punto crítico en el intervalo es $x = \dfrac{\frac{\pi}{2} - 1}{3} \approx 0.190265...$

Comparación: $M = \text{Máx}\{|f'''(0)|, |f'''\left(\dfrac{\frac{\pi}{2}-1}{3}\right)|, |f'''(1)|\} = 27$. Así,

$$|f(0.71) - P(0.71)| \leq 0.194575$$

2.28

$$|f(0.25) - P(0.25)| \leq \left| \frac{M}{3!}(0.25-0)(0.25-1/6)(0.25-1/3) \right|$$

$f^{(3)}(x) = 27\,\text{sen}(1+3x), f^{(4)}(x) = 81\cos(1+3x)$. Punto crítico en $[0,1/3]$, $x = 0.190265$. Entonces $M = 27$ y

$$|f(0.25) - P(0.25)| \leq 0.0078125$$

2.29 Aplicar la fórmula para el error general en interpolación cúbica $h = \pi/2$.

2.30 Aplicar la fórmula para el error general en interpolación cúbica con $h = 0.2$

2.34

$$|f(x) - P_n(x)| \leq \frac{M}{(n+1)!}|(x^* - x_0)(x^* - x_1)\cdots(x^* - x_n)| \leq \frac{M}{(n+1)!}(b-a)^{n+1}$$

pues $|(x^* - x_i)| \leq (b-a)$ para cada $i = 0,1,...,n$.

2.35

$$\begin{cases} S_0(x) = 0 - 1/2(x-0) + 0(x-0)^2 + 3/2(x-0)^3 & \text{si} \quad x \in [0,1], \\ S_1(x) = 1 + 4(x-1) + 9/2(x-1)^2 - 3/2(x-1)^3 & \text{si} \quad x \in [1,2]. \end{cases}$$

2.36.a

$$\begin{cases} S_0(x) = -160 + 1.49061(x-100) + 0 \cdot (x-100)^2 - 0.00002(x-100)^3 & \text{si} \quad x \in [100,200], \\ S_1(x) = -35 + 0.76876(x-200) - 0.00721(x-200)^2 + 0.00002(x-200)^3 & \text{si} \quad x \in [200,300], \\ S_2(x) = -4.2 + 0.10832(x-300) + 0.0006(x-300)^2 - 3.77272 \times 10^{-6}(x-300)^3 & \text{si} \quad x \in [300,400], \\ S_3(x) = 9 + 0.117952153(x-400) - 0.0005(x-400)^2 + 1.28229 \times 10^{-6}(x-400)^3 & \text{si} \quad x \in [400,500]. \\ S_4(x) = 16.9 + 0.05287(x-500) - 0.0001(x-500)^2 + 4.43540 \times 10^{-7}(x-500)^3 & \text{si} \quad x \in [400,500]. \end{cases}$$

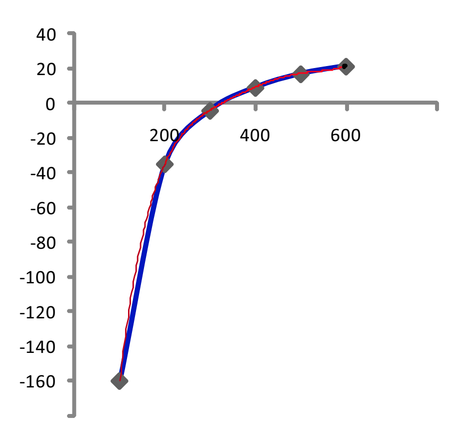


Figura 5.1 El polinomio interpolante es la gráfica en roja, en azul está el trazador cúbico.

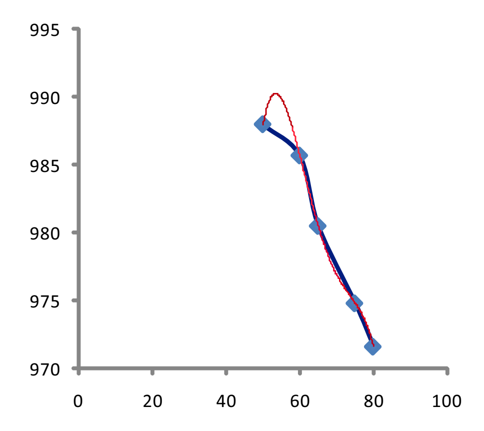


Figura 5.2 El polinomio interpolante es la gráfica en roja, en azul está el trazador cúbico.

2.36.b $S_3(450) = 13.8079(cm^3/mol)$

2.36.c

2.37

Soluciones del Capítulo 3

Soluciones del Capítulo 4

4.1.a

$$\lim_{n\to\infty}\left|\frac{x_{n+1}+2}{(x_n+2)^q}\right| = \lim_{n\to\infty}\left|\frac{2(x_n+2)^{2-q}}{(3x_n+3)}\right|,$$

Así, si $q=1$ el límite es 0, si $q=2$ el límite es $1/3$ y si $q>2$, el límite es ∞.

4.40 Si $\varepsilon>0$ y $1-\varepsilon>0$ entonces $m=3(1-\varepsilon)^2$ y $M=6(1+\varepsilon)$. Necesitamos $x_0\in[1-\varepsilon,1+\varepsilon]$ y además $|x_0-1|<2m/M$.
Como $|x_0-1|<\varepsilon$ bastaría con que $\varepsilon<2m/M \implies 0<\varepsilon<1/3$.

4.77 El máximo de $G(s,t) = s/t^2$ en $I_\varepsilon \times I_\varepsilon$ es $M_\varepsilon = \dfrac{(1+\varepsilon)}{(1-\varepsilon)^2}$ si $1-\varepsilon > 0$.

$\varepsilon \cdot M_\varepsilon < 1 \implies \varepsilon + 2\varepsilon < 1 \implies \varepsilon < 1/3$. Luego, si $x_0 \neq x_1$ están en $I_\varepsilon =]2/3, 4/3[$, tendríamos la convergencia asegurada.

Soluciones del Capítulo 5

5.12.b $n \geq \sqrt[4]{\dfrac{\pi^5}{180\delta}}$.